Online Exploration for Detecting Shifts in Fresh Intent

Damien Lefortier[1,2]
damien@yandex-team.ru

Pavel Serdyukov[1]
pavser@yandex-team.ru

Maarten de Rijke[2]
derijke@uva.nl

[1]Yandex, Moscow, Russia
[2]University of Amsterdam, Amsterdam, The Netherlands

ABSTRACT

In web search, *recency ranking* refers to the task of ranking documents while taking into account freshness as one of the criteria of their relevance. There are two approaches to recency ranking. One focuses on extending existing learning to rank algorithms to optimize for both freshness and relevance. The other relies on an aggregated search strategy: a (dedicated) *fresh vertical* is used and fresh results from this vertical are subsequently integrated into the search engine result page. In this paper, we adopt the second strategy. In particular, we focus on the fresh vertical prediction task for repeating queries and identify the following novel algorithmic problem: how to quickly correct fresh intent detection mistakes made by a state-of-the-art fresh intent detector, which erroneously detected or missed a fresh intent shift upwards for a particular repeating query (i.e., a change in the degree to which the query has a fresh intent). We propose a method for solving this problem. We use *online exploration* at the early start of what we believe to be a detected intent shift. Based on this exploratory phase, we correct fresh intent detection mistakes made by a state-of-that-art fresh intent detector for queries, whose fresh intent has shifted. Using query logs of Yandex, we demonstrate that our methods allow us to significantly improve the speed and quality of the detection of fresh intent shifts.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Recency ranking; aggregated search; vertical search; intent shift

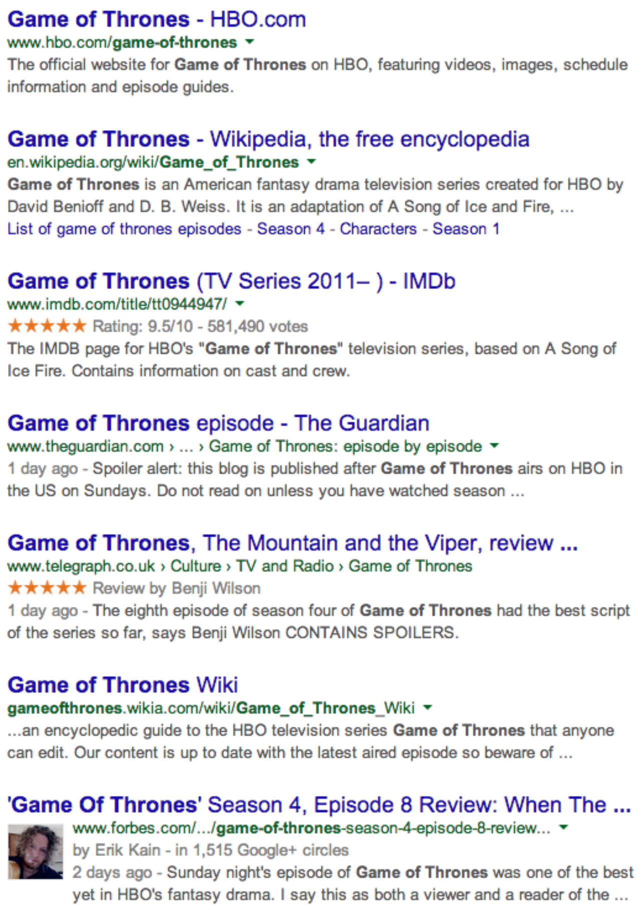
1. INTRODUCTION

In web search, *recency ranking* refers to the task of ranking documents while taking into account freshness as one of the criteria of their relevance [5, 8, 11]. This is particularly important for *recency sensitive queries*, i.e., queries for which a user expects topically relevant documents to be also fresh (that is not older than a few days), and failing to recognize the temporal aspect of such queries will likely lead to user dissatisfaction. Previous work on recency ranking comes in two broad flavours: one type of approach focuses on improving the ranking of generic web results by extending existing learning to rank algorithms to optimize for both freshness and relevance using temporal features [5, 8, 11]. The second type of approach to recency ranking is an aggregated search based approach in which a dedicated *fresh vertical* (indexing only fresh documents) is used, whose results are integrated into the search engine result page (SERP), once the need in fresh content (fresh intent) has been detected for a particular query with some probability [29] (see §5 for related work). In this paper, we adopt and contribute to the latter approach to recency ranking.

Importantly, in the aggregated search-based approach to recency ranking approaches, one does not always integrate fresh results amongst generic web results by grouping them into slots as is typically done in aggregated search with other types of verticals [21]. Instead, fresh results are often mixed together with generic web results—this seems to have become the default (e.g., at Google and Bing, as of June 2014). Consider, e.g., the query "Game of Thrones," whose results from Google are shown in Fig. 1: importantly, some documents of this SERP were published only a few hours before the query was issued and the page is a mixture of both fresh and generally relevant documents.

Within the aggregated search-based approach to recency ranking, we need to decide when and how to integrate fresh results on the SERP. To this end, we can rely on existing research on aggregated search. This task is usually decomposed into two subtasks: *vertical selection* [1, 2] and *result merging* [3, 25]. We focus on the vertical selection task and, as we will show in §2.2, a special treatment is required for this task in the case of a fresh vertical. The vertical selection task depends on assessing what we call the *fresh intent* of a query: the probability that a user issuing the query to the search engine is looking for fresh content. This task is similar to, but different from, assessing the newsworthiness of a query when working with a news vertical [9, 11, 20]: fresh intent is not restricted to news events and also concerns queries about, e.g., the last episode of a television series for which fresh results are required (see Fig. 1) but which do not make the news most of the time. To tackle this task, we use a state-of-the-art *fresh intent detector* that exploits a large and diverse set of features (see §2.1).

The predicted fresh intent from this detector is then used to determine, for each query, the number of fresh results (if any) to integrate on the SERP as well as their position. To give some examples, our recency ranking system at Yandex (Russia's leading search engine) uses the predicted fresh intent value 0.10 as a uniform threshold for determining which queries are predicted as *fresh* at a given point in time, that is, for which fresh results have a chance to be integrated somewhere into the SERP. A similar threshold was used in related work, e.g., in [9]. In the production environment of Yandex, the value 0.10 corresponds, on average, to 1 fresh document at



Game of Thrones - HBO.com
www.hbo.com/game-of-thrones
The official website for Game of Thrones on HBO, featuring videos, images, schedule information and episode guides.

Game of Thrones - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Game_of_Thrones
Game of Thrones is an American fantasy drama television series created for HBO by David Benioff and D. B. Weiss. It is an adaptation of A Song of Ice and Fire, ...
List of game of thrones episodes - Season 4 - Characters - Season 1

Game of Thrones (TV Series 2011–) - IMDb
www.imdb.com/title/tt0944947/
★★★★★ Rating: 9.5/10 - 581,490 votes
The IMDB page for HBO's "Game of Thrones" television series, based on A Song of Ice Fire. Contains information on cast and crew.

Game of Thrones episode - The Guardian
www.theguardian.com › ... › Game of Thrones: episode by episode
1 day ago - Spoiler alert: this blog is published after Game of Thrones airs on HBO in the US on Sundays. Do not read on unless you have watched season ...

Game of Thrones, The Mountain and the Viper, review ...
www.telegraph.co.uk › Culture › TV and Radio › Game of Thrones
★★★★★ Review by Benji Wilson
1 day ago - The eighth episode of season four of Game of Thrones had the best script of the series so far, says Benji Wilson CONTAINS SPOILERS.

Game of Thrones Wiki
gameofthrones.wikia.com/wiki/Game_of_Thrones_Wiki
...an encyclopedic guide to the HBO television series Game of Thrones that anyone can edit. Our content is up to date with the latest aired episode so beware of ...

'Game Of Thrones' Season 4, Episode 8 Review: When The ...
www.forbes.com/.../game-of-thrones-season-4-episode-8-review...
by Erik Kain - in 1,515 Google+ circles
2 days ago - Sunday night's episode of Game of Thrones was one of the best yet in HBO's fantasy drama. I say this as both a viewer and a reader of the ...

Figure 1: Google results for the query "Game of Thrones," with fresh results on top.

the 10th position of the SERP, while, for example, the value 0.35 corresponds, on average, to 3–4 fresh documents spread across the SERP, and the value 0.60 corresponds, on average, to a SERP with 5–6 fresh documents and with a majority of fresh documents in the top of the SERP. Of course, this is subject to the choice of a particular search engine and depends on its mechanism used to integrate fresh results into SERPs, which is beyond the scope of this paper.

The main challenge associated with accurately predicting the fresh intent of a query is that for many recurring queries the fresh intent may change over time and that such *intent shifts* can happen really fast. We illustrate this in Fig. 2 for a query about a government department that made the news. There, we see that a large and sudden intent shift upwards occurred at the 18th hour, when the real fresh intent went up from 0.06 to 0.28 within a single hour (see the green curve, which plots the real fresh intent of this query).[1]

Failing to properly integrate relevant fresh results in the SERP for fresh queries usually happens due to incorrect fresh intent detection and is likely to lead to user dissatisfaction. The problem is illustrated in Fig. 2 (see the blue curve, which plots the predicted fresh intent of this query—which clear lags behind the real fresh intent following the fresh intent shift). We see that the potential dissatisfaction with the search results, which we assume to be proportional to the difference between the real and predicted fresh intent of a query, can be important. As we see, fresh results could be inadequately integrated into the SERP due to the incorrectly pre-

[1]We use surrogate ground truth for fresh intent for defining the real fresh intent; we discuss and explain the need for using this type of ground truth, and demonstrate its extremely high quality, in §2.2.

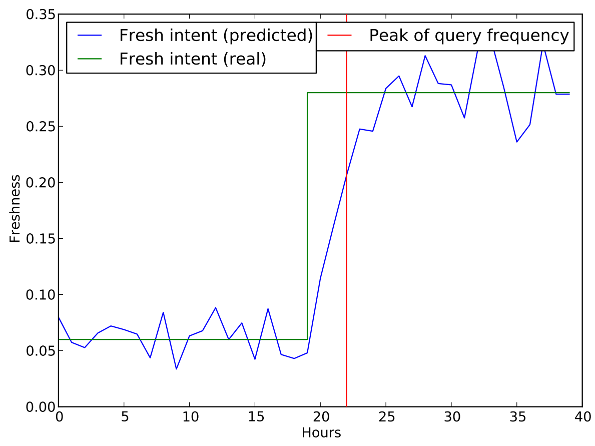


Figure 2: The evolution of real and predicted fresh intent for a random query, which quickly became fresh.

dicted fresh intent of the query around, and at, the peak of user interest (measured as the number of queries per hour) following the fresh intent shift that happened three hours before the peak.

In §2.2, we will show that a state-of-the-art fresh intent detector, although highly precise on average, may fail to detect shifts upwards in fresh intent soon after such a shift occurred in user preferences but is not yet well reflected in the available data sources, and that it may, in fact, need a long time to properly detect the real fresh intent of a query (as in Fig. 2). Therefore, the core challenge of the present study is to learn to correct for such fresh intent detection mistakes and to rapidly recognize large changes in the fresh intent of a query when the detector is failing.[2] This is a challenging problem, because, during or immediately after changes in the fresh intent of a query take place, we usually do not have much information yet to determine the exact fresh intent of a query—that is exactly why mistakes happen. To the best of our knowledge, this important problem has not been studied in previous work.

Our idea of solving this problem of detecting shifts in fresh intent of a query is to perform *online exploration* at the very beginning, when we are starting to detect an increase in fresh intent, but when we are still uncertain that the detected increase is really due to a shift that took place, or a shift that is about to happen, in user preferences, or due to the noise in our observations. We integrate fresh results on the SERP for a query that has displayed an increase in fresh intent and gather user feedback to quickly correct our predicted fresh intent for queries whose fresh intent has shifted and that were not handled well by our fresh intent detector. Obviously, this procedure cannot be followed for all queries for which the predicted fresh intent has slightly increased due to the risk of degrading the search experience by presenting fresh results at the top of too many SERPs, whilst most users actually do not need that. Hence, we learn to select only the most appropriate queries for this treatment and we do so using a semi-supervised machine learning model trained to predict the probability for the fresh intent of a query to have actually shifted considerably after we started to detect a small increase in its predicted fresh intent.

The main contributions of this paper are the following.

- We present a large scale analysis of how a state-of-the-art

[2]We focus only on fresh intent shifts upwards, because not showing any relevant fresh results at the peak of user interest here is much more critical that showing fresh results a bit too long, i.e., when people care less, which might happen with potential detection mistakes of fresh intent shifts downwards.

fresh intent detector deals with fresh intent shifts upwards using query logs of Yandex. Our observations motivate the following novel algorithmic problem: how to quickly correct mistakes in fresh intent detection made by this detector after an instant intent shift.

- Using a semi-supervised learning approach we demonstrate how to better predict whether the fresh intent of a query has actually substantially shifted upwards after a small observed increase in its fresh intent. We build on this prediction to decide whether to start an online exploration in order to refine our estimate of the query's fresh intent.
- We propose and evaluate different methods for rapid online detection of shifts in fresh intent using online exploration. We find that our methods allow us to significantly improve the speed and the quality of the detection of intent shifts without deteriorating overall retrieval performance.

In §2, we motivate our algorithmic problem through a presentation of a fresh intent detector and an analysis of intent shifts in the wild. We describe our machine learning model for predicting the probability for the fresh intent of a query to have shifted after we started to detect an increase in its predicted fresh intent in §3.1. Methods for performing online exploration are discussed in §3.2. We present our results in §4; §5 is devoted to related work; we conclude in §6.

2. MOTIVATION

In this section, we motivate our novel algorithmic problem, that is how to quickly correct fresh intent detection mistakes made by a state-of-the-art fresh intent detector for queries whose fresh intent has shifted, through a presentation of our fresh intent detector (§2.1) and an analysis of intent shifts "in the wild" (§2.2).

2.1 Fresh intent detector

We describe our state-of-the-art *fresh intent detector*, that is, our model for performing vertical selection, which estimates the fresh intent of a query as discussed in §1. Recall that in the aggregated search-based approach to recency ranking which we follow, the predicted fresh intent of a query is used to determine the number of fresh results to be integrated on the SERP and their position.

The most common approach to estimating the fresh intent of a query [9, 11, 20, 29] is to combine features extracted both from search engine query logs (representing the fresh content demand), as well as from the fresh vertical (representing the fresh content production). The idea is that a combined increase in demand and production of fresh content related to the query topic most probably indicates an increase in the fresh intent of the query.

Our fresh intent detector is based on a machine learned model that is trained offline using assessments from professional judges, hired by Yandex. The data used for training and testing this model was collected according to the following procedure. We sampled 10,000 queries over a period of 3 months with upsampling in order to have enough queries with a high fresh intent. Each query was then labelled by at least 2 judges, who were asked to determine the fresh intent of each query, that is, to determine to which extent these queries express an interest in upcoming or on-going events for which users would prefer fresh content at the time of the issue (without taking into account the relevance of search results). For each query, judges were asked to decide in which interval the query's fresh intent is and we use the mean values of these intervals as our targets for learning. We train our fresh intent detection model by minimizing the mean squared error of the difference between the predicted and real fresh intent for labeled queries in the training data using Gradient Boosted Regression Trees (GBRT) [13]. This approach was used also in [29].

The features used in our model (about 100 overall) are borrowed from existing studies, and, most notably, include the following:

- Features extracted from our fresh vertical (representing the fresh content production) similar to the ones described and discussed in existing studies on query recency sensitivity and newsworthiness detection [9, 11, 20]. Well-performing examples include the number of fresh documents retrieved by the query in our fresh vertical (NewDocsTotal), or the ratio of NewDocsTotal and the number of web results returned by Yandex for the query (NewDocsFraction), or the fraction of fresh documents added to the fresh vertical in the last X hours containing all the query words (HasAllWordsXh), or also the probability for the query to be generated by language models from our fresh vertical.
- Features extracted from the query stream (representing the fresh content demand) similar to the ones described and discussed in previous work [9, 11, 20]. Well-performing examples include the number of issues of the query over different time periods or the ratio of the number of issues of the query in the last day and in the last week (QueryConstrast), or also the probability for the query to be generated by language models from the query stream.
- Query features such as the number of words or boolean features indicating whether the query is navigational, about a product, etc. [20].
- Features extracted from social data streams (e.g., Twitter), such as those described in [12, 20].
- Click-through features, that is, the CTR of fresh results on the SERP. Such features were discussed in [26].

These features are continuously updated in near real-time. Hence, potential time delays in accurate prediction of the fresh intent of queries whose fresh intent has suddenly shifted are not due to time-consuming system processes, but because the signal indicating a change in fresh intent is sometimes weak and intent shift can therefore only be detected with high uncertainty.

To estimate the performance of the fresh intent detector, we use the average square error, which is around 0.025 as estimated using 10-fold cross validation, thus proving the high quality of our fresh intent detector. The square error can be very large (up to 0.48) for some queries, which means that our detector can substantially fail for some queries as we will see in the next section.

2.2 Fresh intent shifts in the wild

We investigate how the state-of-the-art fresh intent detector (as described in §2.1) deals with fresh intent shifts upwards. More precisely, we investigate the following questions through an analysis of the query logs of Yandex.

RQ1 How fast is the state-of-the-art fresh intent detector defined in §2.1 able to accurately estimate the fresh intent of queries, whose fresh intent has shifted, right after the shift has happened because of some external event?

RQ2 What is the cost of the error in our prediction of fresh intent during a certain period of time, when we are catching up with the correct detection of the intent shift?

Surrogate ground truth. We start by presenting the surrogate ground truth for the fresh intent of queries. In order to understand how our fresh intent detector deals with fresh intent shifts upwards, we need to compare our predicted fresh intent to a ground truth. Due to the massive amount of data to analyze we cannot rely on expensive manual assessments and, instead, we propose a method for generating this ground truth based on two assumptions.

The first assumption is that the predicted fresh intent, once stabilized at a new (higher) fresh intent value corresponds to the real fresh intent, because the state-of-the-art fresh intent detector is highly precise once we have enough signal (see §2.1). Second, we assume that after an intent shift has occurred, as confirmed by the stabilized prediction produced by our fresh intent detector, we can look back into the past to find the moment in time when our predicted fresh intent, yet unstable, started to increase from its previous stable value eventually reaching a new stable value. We assume this point to be the moment of the intent shift, since we know, *post factum*, that this increase, because it was followed by a sustained growth, was not due to noise in our observations but due to an actual shift in user preferences, as happened, e.g., at the 18^{th} hour in Fig. 2. These two assumptions behind our surrogate ground truth are very strong and we checked that they hold as follows.

Let t_1^q be the point in time of the intent shift of a query q (e.g., the 18^{th} hour in Fig. 2) and let t_2^q be the point in time when the predicted fresh intent stabilized (e.g., the 26^{th} hour in Fig. 2).[3] The latter is the moment in time from which we are able to infer the real fresh intent as discussed above. We randomly selected 20 queries whose fresh intent has shifted, from 20 randomly selected days in February 2014, so 400 queries overall, and checked that the real fresh intent at t_1^q is indeed equal to the predicted fresh intent at t_2^q using manual assessments (see §2.1). This was the case for 85% of the queries and we observed that t_1^q is really close (within minutes) to the beginning of the real-world event that triggered the intent shift (e.g., a train accident or the release of a new episode of a television series), thus confirming our two assumptions. For the 15% for which this is not the case, we observe that the problem is due to noise in our observations that resembles a fresh intent shift (note that these queries are uniformly distributed in our data in terms of number of issues per day).

Algorithm 1 allows us to retrospectively detect pairs t_1^q and t_2^q in the data and thus generate our surrogate ground truth for the real fresh intent between t_1^q and t_2^q. This algorithm works as follows. First, we assume that we are starting to detect an increase in real fresh intent after observing a certain increase of the median predicted fresh intent over the last N issues of the query in a certain time window. In our study, we consider an increase of at least 0.02 to be important; this threshold allows us to detect intent shifts really fast, e.g., before the peak of user interest for a query, as illustrated in Fig. 2. We empirically set the size of the time window, used here as well as further in the paper, to six hours; we obtained similar results using other values. We use the boolean function `Inc`, which returns true when an increase is detected, which means that the following condition became true:

$$\mu_{current} - \mu_{initial} > 0.02, \quad (1)$$

where *initial* (*current*) is the sequence of predicted fresh intent values for the first (last) N issues of q in our time window, and where μ is the median value of each sequence. Second, we consider that the predicted fresh intent of a query has stabilized when it has reached a value at most 0.02 less than the real fresh intent (according to our surrogate ground truth). We use the boolean function `End`, which returns true when the predicted fresh intent has stabilized, which means that the following condition became true:

$$\mu_{real} - \mu_{current} < 0.02 \quad (2)$$

[3]In the next paragraph we explain how to compute these two points in time.

Algorithm 1: Estimation of t_1^q and t_2^q

```
input  : List $I_q$ of issues of a query $q$ in a time window
output: $t_1^q$, $t_2^q$, fresh intent reached at times $t_1^q$ and $t_2^q$
// To mark found increase in fresh intent
FoundInc ⟵ $False$
$t_1$ ⟵ 0 $t_2$ ⟵ 0
for $i$=$N$ to $length(I_q) - 1$ do
    // Get median fresh intent over first N
       issues of $q$ in a 6-h window (past).
    InitialFresh IntentSeq$_N$ ⟵ GetFresh IntentFst$_N(I_q, i)$
    // Get median fresh intent over last N
       issues of $q$ in the same window.
    FreshIntentSeq$_N$ ⟵ GetFreshIntentLst$_N(I_q, i)$
    // Looking for increase in fresh intent
    if $FoundInc$ == $False$ then
        if $Inc(InitialFreshIntentSeq_N, FreshIntentSeq_N)$ then
            FoundInc ⟵ $True$, $t_1$ ⟵ GetCurrentTime()
    // Growth ended
    else if $End(FreshIntentSeq_N)$ then
        return $t_1$,GetCurrentTime(), FreshIntentSeq$_N$
    // Growth continues
    else
        // Nothing
// Stable or still growing fresh intent
return NoIntentShift
```

where μ_{real} is the real fresh intent, that is the largest median fresh intent over N issues reached by q between t_1^q and the moment when fresh intent starts to decay.

Cost of prediction error. Next, we formalize the cost $c_q(t_1, t_2)$ of the prediction error of the real fresh intent of a query q in a given time window $[t_1, t_2]$:

$$c_q(t_1, t_2) = \int_{t_1}^{t_2} f_q(t) \cdot |\text{fresh\_intent}_q(t) - \widehat{\text{fresh\_intent}}_q(t)| \, \mathrm{d}t \quad (3)$$

where $\text{fresh\_intent}_q(t)$ and $\widehat{\text{fresh\_intent}}_q(t)$ are, respectively, the real and predicted fresh intent of q at time t, and $f_q(t)$ is the frequency of q at time t.

For example, for the query shown in Fig. 2, the cost, as defined above, corresponds to the area between the green and blue curves, when we are catching up with our prediction following the intent shift, weighted by the frequency of q at time t. This measure allows us to reliably compare different methods for predicting fresh intent, as a smaller cost does imply a better prediction and vice versa.

Dataset. We created two datasets, D_{train} and D_{test}, using data from query logs of Yandex from respectively 10 consecutive days in February 2014 and 10 consecutive days in March 2014. We selected all queries of users from Russia, whose fresh intent started to increase according to Algorithm 1. We empirically set N, the number of consecutive issues of a query used to compute the median in Algorithm 1, to 10 (we obtained similar results for all values between 5 and 20). We filtered out queries that were issued 20 times or less in the last 6 hours thus focusing on recurring queries as less frequent queries are not of particular interest for us here. Then, we use our surrogate ground truth for fresh intent to determine the queries whose fresh intent has indeed shifted. We consider an increase of at least 0.10 in less than one day (between t_1^q and t_2^q) to

Table 1: Time delays (TD) and cost over all queries in the time window, whose fresh intent has shifted.

TD (mean)	TD (median)	TD (skew)	Cost
7.9 h	9.5 h	0.64	37.66

be important. In this way, we obtain a fully labelled dataset that we will use for analyzing how the state-of-the-art fresh intent detector deals with fresh intent shifts and also for training and testing our method for solving the problem at hand.

Specifically, we obtained $3,402$ and $5,054$ queries for D_{train} and D_{test}, respectively, whose fresh intent has shifted, and $83,711$ and $72,186$ queries for which we started to detect a small increase in their predicted fresh intent but not due to an intent shift.

Results. Here, we use D_{train} and consider only queries whose fresh intent has indeed shifted. We observe important time delays with a median delay of about 9.5 hours, as shown in Table 1. Such important delays are due to the fact that, right after an intent shift happened (due to some external event), only a few queries were issued and little relevant content about this event was created and indexed yet (as in Fig. 2), which made the signal weak and the predicted fresh intent uncertain.

About 1% of queries (issues) out of all fresh queries (issues) and 2–4% of queries (issues) out of all fresh queries (issues) whose fresh intent is higher than 0.35 are affected by time delays related to fresh intent shifts. This is a substantial number considering that failing to show fresh results in time can lead to important user dissatisfaction and might even lead to people switch to another search engine to find the information they are looking for—a big failure for the search engine in the long term considering the huge amount of traffic every day.[4] Here, in D_{train}, more than 1 million query issues are affected. The overall cost is 37.66. This value is going to be useful in §4 when weThe overall cost is 37.66. This value is going to be useful in §4 when we study different exploration strategies and their usefulness for predicting fresh intent.

The overall cost is 37.66. This value is going to be useful in §4 when we study different exploration strategies and their usefulness for predicting fresh intent.

Discussion. We can now answer our research questions. The answer to RQ1 is that the state-of-the-art fresh intent detector often makes mistakes soon after an intent shift occurred and may need a long time to properly detect the real fresh intent of a query with time delays of 9.5 hours on average. The answer to RQ2 is that the cost of the error in our prediction is substantial as indicated by the large percentage of queries for which fresh results could be not adequately integrated into the SERP due to the incorrectly predicted fresh intent of the query.

These observations motivate the following algorithmic problem: *how to quickly correct fresh intent detection mistakes made by our fresh intent detector after an instant intent shift upwards?* To avoid missing the peak of user interest, this correction has to be made as soon as we have started to detect an increase in fresh intent. Small spikes in predicted fresh intent happen for a large number of queries every day, and determining the exact fresh intent right after those small increases is thus a challenging task.

3. METHOD

Next, we present our method to quickly correct our predicted fresh intent for queries whose fresh intent has shifted upwards. Our proposed solution is two-fold. First, we estimate the probability for the fresh intent of a query to have shifted considerably after we detected a small increase in its predicted fresh intent (§3.1). Second, we select queries depending and this probability and explore those queries' fresh intent online by gathering user feedback to test the hypothesis made at the previous step (§3.2).

[4] http://www.internetlivestats.com/google-search-statistics/#trend

3.1 Predicting intent shift

We investigate how to predict the probability of the fresh intent of a query to have shifted after we have detected an increase in its predicted fresh intent: $\Pr(\text{IntentShifted}_q)$. It is important to distinguish this prediction task from the fresh intent estimation task described in §2.1. We use a state-of-the-art learning method to learn a model for predicting $\Pr(\text{IntentShifted}_q)$. An underlying question is whether we can learn this probability well enough, so that online exploration can be effective. The latter means that we need to achieve high recall (to get most queries whose fresh intent has shifted as candidates) at sufficiently high precision (so as not to "pay" too much on the possible degradation resulting from exploring other queries).

Dataset. There is a trade-off between how quickly we will be able to predict $\Pr(\text{IntentShifted}_q)$ and the complexity of the learning task. If we use a small threshold for defining when the predicted fresh intent started to increase, we will have a heavily imbalanced data set with more negative examples (i.e., queries whose fresh intent did not actually shift despite that slight increase). On the other hand, if we use a higher threshold, we will wait for a stronger increase before exploring, so we will wait longer before detecting fresh intent shifts. Here, we will use our datasets D_{train} and D_{test} (see §2.2), which are imbalanced, but we will see that learning is, nevertheless, possible.

Learning & features. We use GBRT [13] along with a logistic transformation to learn a model for predicting $\Pr(\text{IntentShifted}_q)$ using D_{train}. These are the features for a given query q; the last issues of q are all the issues made in the last 6 hours.

- IntentMax, IntentMin, IntentMean, IntentStd, IntentSkew: the maximum, minimum, mean, standard deviation and skew of the predicted fresh intent for the last issues of q.
- IssueMax, IssueMin, IssueMean, IssueStd, IssueSkew, which are the maximum, minimum, mean, standard deviation and skew of the time passed since each of the last issues of q.
- The number of issues of q in the last 6 hours.
- The Kendall tau rank correlation coefficient between the time passed since each of the last issues of q and the predicted fresh intent of each issue. This feature gives some information about the shape of the small increase in q's predicted fresh intent that we have detected.
- All the features described in §2.1.

We explored the usage of up/down sampling but this did not significantly improve the performance of our model.

Performance. In Fig. 3, we plot the performance of our predictive model for different precision and recall values. We are able to learn our model with reasonable accuracy despite the imbalanced nature of our data set. E.g., we are able to accurately predict 92% of the queries that became highly fresh at 20% precision level. This result outperforms random guessing for which queries fresh intent has shifted (we have a fixed precision of 7% in this case). Below, we will see that the performance of our predictive model is good enough for the subsequent exploration phase to be effective.

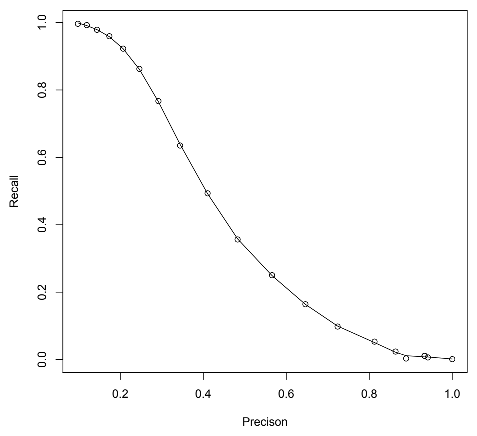


Figure 3: Recall versus precision of our model for predicting the probability for the fresh intent of a query to have shifted after we started to detect an increase in its predicted fresh intent.

Table 2: Top 10 features according to their contribution to our model for predicting the probability for the fresh intent of a query to have shifted after we have started to detect an increase in its predicted fresh intent.

Rank	Feature	Score	Rank	Feature	Score
1	IntentSkew	7.28	6	NewDocsFraction	3.03
2	ShowProb	5.80	7	HasAllWords5h	2.76
3	IntentMax	4.65	8	IssueMean	2.75
4	IntentMedian	3.75	9	QueryFreq	2.56
5	IntentStd	3.18	10	IssueStd	2.21

We turn to the contribution of individual features. The top 10 features according to their weighted contribution to our model are shown in Table 2; see [15, §10.13] for a description of the weights. The Intent and Issue features perform well; other well-performing features are the probability for fresh results to be integrated on the SERP aggregated over each word of the query (ShowProb), NewDocsFraction (see §2.1), HasAllWords5h (§2.1), and the query frequency (QueryFreq).

The features that we added on top of the ones used in our fresh intent detector (as detailed in §2.1) allow us to notably improve the performance: about 5% recall at each precision level. However, the main contribution of this section is about learning our model on a new target (our surrogate ground truth, i.e., not the current, but the future predicted fresh intent). This new predictive model also opens the opportunity to test new real-time features, which would not be very strong in the default fresh intent detector, but which could yield really good results in our model here.

Discussion. Using the model described in this section, we can predict the probability for the fresh intent of a query to have shifted after we have detected an increase in its predicted fresh intent, that we denoted $\Pr(\text{IntentShifted}_q)$. There is a trade-off when choosing the precision level at which to use our model, between how many intent shifts we will be able to detect (recall) versus how many queries will (possibly) be degraded while exploring (precision); see Fig. 3. In §4, we discuss how we tune, in order to maximize the retrieval performance, this precision level—based on which we obtain a probability threshold p to be used on the outcome of our predictive model to select queries.

3.2 Online exploration

We present our algorithms for performing online exploration for queries selected by the predictive model introduced in §3.1. This will allow us to gather user feedback to quickly correct our predicted fresh intent for queries, whose fresh intent has shifted, right after we started to detect an increase in their predicted fresh intent.

Note that each query, whose fresh intent has shifted, was not necessarily issued thousands of times before our fresh intent detector catches up with the intent shift (with a median of 81 times and a mean of 281 times in D_{test}). We therefore do not have much time to gather user feedback for solving the task at hand. We need to keep in mind this interesting aspect of our setting when designing our exploration algorithms.

Online exploration can be done in different ways and we need to decide how to proceed in order to satisfy the following goals: (1) To correct our predicted fresh intent for queries, whose fresh intent has shifted as fast as possible in order to reduce the cost from §2.2, while (2) limiting the risk of degrading the search experience while exploring. We propose two exploration methods, *multi-armed bandits* and *explore on top*.

3.2.1 Multi-armed bandits

The state-of-the-art in online learning would suggest to model our online exploration problem as a multi-armed bandit problem (MAB) [4, 6], which is typically used to model situations where it is possible to explore and exploit simultaneously, as has been done previously for creating a unified search federation system [19, see §5], and for ranker evaluation [18]. We do so in the following algorithm, that we call *Bandits*.

Our formalization of the task at hand as a bandit problem runs as follows. We consider the SERP as an arm (or action) [19] and consider only actions that integrate fresh results on the SERP differently. Each action therefore corresponds to deciding how many fresh results to integrate on the SERP as well as their position. In other words, each action is associated with a fresh intent value (the one to use to obtain this SERP) and we can thus, by finding the best action, correct our predicted fresh intent for queries whose fresh intent has shifted—thus solving the task at hand.

As we do not use slots for placing fresh results on the SERP, we have 2^N possible actions, where N is the number of fresh and generic web results to mix (usually, 10). This number is too large to be practical, but we can easily limit the number of actions to any number B. To this end, we first define a probability distribution over a query's fresh intent that we discretize by defining B buckets of equal probability. Then, for each action i (from 0 to $B-1$), we define the SERP obtained by integrating fresh results according to the mean fresh intent value of each bucket i. For example, using a uniform probability distribution over a query's fresh intent and 10 buckets, we end up with the following buckets: $[0.0, 0.1]$, $[0.1, 0.2]$, … up to $[0.9, 1.0]$. Each action, then, corresponds to the SERP associated, on average, with the mean (fresh intent value) of each bucket, that is in this case, for each bucket, with 0.05, 0.15, … up to 0.95.

Then, we need to define the reward of each action. We follow [19] and define a reward of an action to be the sum of the rewards of the items composing the SERP and we use a click-skip per-item reward. Specifically, the reward of an action is equal to the sum of the number of clicked documents minus the number of skipped documents. To solve the MAB problem, we propose to use Thompson sampling, a state-of-the-art heuristic for choosing actions [7].

3.2.2 Integration on top and boosting

We also propose the following simple and fast algorithm, specifically designed for solving the task at hand. In this algorithm, that we call *ExploreOnTop*, every time we receive a query to explore, we integrate, for some time, fresh results on top of its SERP and gather user feedback. *ExploreOnTop* integrates *a single* fresh result at the top position of the SERP, thus allowing us to get user feedback that is not too noisy. We propose two exploration methods for deciding for how long we need to integrate this one fresh result at the top position of the SERP, *uniform* and *dynamic*.

Exploration method 1: Uniform. In this algorithm, that we call *ExploreOnTop-Uniform*, each selected query is explored for all the next M consecutive issues following its selection, where M is a parameter to tune experimentally.

Exploration method 2: Dynamic. In this algorithm, that we call *ExploreOnTop-Dynamic*, each selected query is explored at most for all the next M consecutive query issues and we adapt to the user feedback observed to decide when to stop exploring.

This is different from the previous algorithm where the stopping condition is fixed and depends neither on the user feedback observed during the exploration and, therefore, nor on our belief that we gathered enough feedback in order to properly correct our predicted fresh intent for queries whose fresh intent has shifted.

Each time we show a fresh result at the top position of the SERP, we either observe a click or not. We thus have a binomial distribution, whose success probability in each trial is the real click-through rate (CTR) of the fresh result, that we denote FreshCTR_q. We stop when the standard error of the mean of our observations, i.e., of FreshCTR_q, is less than a small value S provided that we explored at least $M' < M$ issues already (for the standard error to be estimated well enough), where S and M' are parameters to tune experimentally. Intuitively, this method allows us to do less exploration on average and therefore to reduce the cost compared to *ExploreOnTop-Uniform*.

After this exploration phase, whether uniform or dynamic, we exploit the user feedback to correct the predicted fresh intent. This can be done either by predicting a new fresh intent value using our fresh intent detector from §2.1 or by using a linear combination of our predicted fresh intent and the observed CTR of the fresh result shown at the top position of the SERP (that we denote $\widehat{\text{FreshCTR}}_q$) as our new prediction of fresh intent as follows:

$$(1-\alpha) * \widehat{\text{fresh\_intent}}_q(t) + \alpha * \frac{\widehat{\text{FreshCTR}}_q}{\hat{R}} \tag{4}$$

Where $\alpha \in [0, 1]$ is a parameter to tune experimentally and $\hat{R}$ (R) is the expected (real) probability of relevance of the fresh result placed at the top position of the SERP (with $\text{fresh\_intent}_q * R = \text{FreshCTR}_q$). This means that we actually *boost* our predicted fresh intent using user feedback. In §4, we use this latter approach.

4. RESULTS

In this section, we investigate how our two-fold method introduced in §3 allows us to quickly correct the predicted fresh intent for queries whose fresh intent has shifted.

4.1 Experimental setup

Research questions. We seek to answer the following research questions.

RQ3 How does the performance of our *multi-armed bandits* approach to online exploration for fresh intent compare with our fresh intent detector?

RQ4 How does the performance of our *explore on top* approach to online exploration for fresh intent compare with our fresh intent detector?

To answer these questions, we perform a comparison of our algorithms against each other, and against our fresh intent detector from §2.1, i.e., when no exploration is performed.

Simulation. We propose and investigate the following experiment simulating a running system using query logs of Yandex. Simulated production environments have already been used in related studies on query intent detection [9, 10].

Every time a query that was not explored yet is issued, we determine whether we are starting to detect an increase in its predicted fresh intent using Algorithm 1. If this is the case, we use our predictive model from §3.1 to predict the probability for the fresh intent of this query to have shifted. If this probability is more than p (to be discussed later), we start to explore this query using one of our exploration approaches (§3.2), i.e., either *Bandits* or *ExploreOnTop*.

In order to simulate a running system, we need to simulate user feedback. To do that, we use the *dependent click model* (DCM) [14], which effectively simulates user feedback, as done in previous studies on online learning to rank [16, 18] and online evaluation [17]. DCM assumes a cascade model of user behavior on the SERP: the user scans the page from top to bottom clicking on results perceived to be relevant and can stop searching after a click [14].

To experiment with *Bandits*, we need to assess the relevance of the top 10 generic web and fresh results for each query at time t (between t_1^q and t_2^q) for DCM. As this data, which is quite difficult to collect for about 80,000 queries, is not available to us, we use using internal statistics from Yandex to estimate the expected relevance of generic web and fresh results depending on their position in the ranking of each vertical as done in [29].

We assume that the first fresh result to be integrated on the top of the SERP in *ExploreOnTop* has a fixed expected probability of relevance $\hat{R}$ (see (4)) for all queries.[5] We experiment with different values and, in 4.2, we show the impact of this expected value, that is of quality of the other parts of Yandex such as fresh ranking, on our results. In all others experiments, though, we use $\hat{R} = 1$ as this allows us to focus on the fresh intent detection and to understand the benefits of our approach providing that other parts of the system such as fresh ranking are of high quality in accordance with the industry standards.

When using DCM, we define the probability (denoted λ_i in [14]) that the user would like to see more results after a click at position i to be 0.8^i as in [28].

Metrics and significance testing. Our main metric is the difference between the number of *upgraded* and *degraded* SERPs, i.e., respectively, the SERPs for which fresh results were integrated better (worse) than with our default fresh intent detector. Such degradations can happen, for instance, for *Bandits*, while exploring when the selected action integrates too many or too few fresh results (see §3.2.1) and, for *ExploreOnTop*, when we show one fresh result at the top position of the SERP or due to a wrong correction, because the user feedback was not informative enough. This metric, which we compute for all queries, represents the *overall performance* of our system, which must be positive for the benefits of our method to be greater than the exploration cost.

We also use the following metrics for queries, whose intent has shifted: the cost (as defined in (3)) and median time delay (see §2.2). We do not report the median time delay for *Bandits* because, as we never really stop exploring, we never totally catch up with the correct detection of the intent shift.

[5]If this is not the case, then we need to (be able to) predict $\hat{R}$ from (4); this prediction task is beyond the scope of this paper and can be tackled, e.g., by using score normalization [24] or by predicting the quality of the top fresh vertical results (here, top 1) [22].

Table 3: Comparison of our different exploration algorithms with our fresh intent detector. A $^\blacktriangledown$ ($^\vartriangle$) denotes significantly worse (better) performance compared to the best performance (the default fresh intent detector) per metric indicated with bold face (see below for details about each column). Underlining indicates the best result on our main metric.

	Cost	SERPs upgraded / degraded	Performance	Median time delay (seconds)
Fresh Intent detector	37.66$^\blacktriangledown$	–	0$^\blacktriangledown$	26,740$^\blacktriangledown$
Bandits (B=5)	35.48$^{\blacktriangledown\vartriangle}$	1,015,774$^\blacktriangledown$ / 711,947$^\blacktriangledown$	303,827$^{\blacktriangledown\vartriangle}$	–
ExploreOnTop-Uniform (M=11, $\hat{R} = 0.5$)	34.63$^{\blacktriangledown\vartriangle}$	1,026,711$^\blacktriangledown$ / 701,011$^\blacktriangledown$	325,700$^{\blacktriangledown\vartriangle}$	11,777$^{\blacktriangledown\vartriangle}$
ExploreOnTop-Uniform (M=11, $\hat{R} = 1$)	**30.67** $^\vartriangle$	1,279,220 / 448,503	830,716 $^\vartriangle$	**11,057** $^\vartriangle$
ExploreOnTop-Dynamic (M=11, $\hat{R} = 1$)	30.78 $^\vartriangle$	**1,282,529** / **445,197**	<u>**837,330**</u> $^\vartriangle$	11,387 $^\vartriangle$

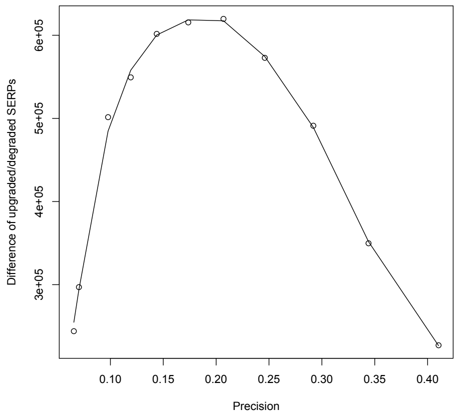


Figure 4: Performance versus precision of our model for predicting the probability for the fresh intent of a query to have shifted after we started to detect an increase in its predicted fresh intent using *ExploreOnTop-Uniform* with $\alpha = 0.04$ and $M = 11$ on D_{train}.

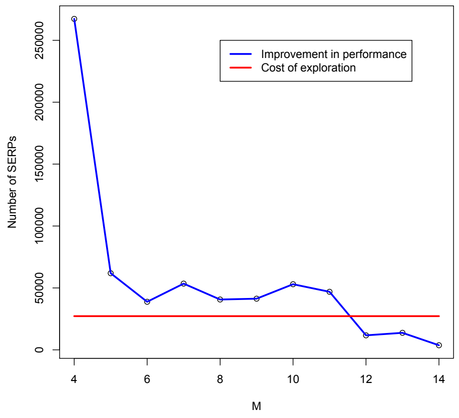


Figure 5: The improvement in performance versus the cost of exploration as M increases using *ExploreOnTop-Uniform* with p set to reach a 20% precision level and $\alpha = 0.04$ on D_{train}.

Statistical significance of observed differences between an algorithm and the best performing algorithm on each metric is tested using a one-tailed paired t-test (at the 0.05 level) using the results from each run.

Settings and parameters. For *ExploreOnTop-Uniform*, we set p, the threshold for deciding whether to explore (see §3.1), in order to reach a 20% precision level, M, the number of consecutive issues to explore to 11, and α, the coefficient used to boost our predicted fresh intent using observed user feedback (see 4) to 0.4—as these parameters allow us to reach the best performance after trying all triplets (p, M, α) with steps (1,1,0.1) on D_{train}. We show, in Fig. 4, the overall performance depending on the precision of our predictive model from §3.1, on Fig. 5, we show that the exploration cost (red) overcomes the improvement in performance (blue) after $M = 11$, and, on Fig. 6, we plot the cost as a function of α, where we see a convex function as expected.

For *ExploreOnTop-Dynamic*, we also need to tune S, the threshold on the standard error of the mean to stop exploring, and M', the minimum number of consecutive issues to explore. The best performance is reached when using the same values for p, M and α as for *ExploreOnTop-Uniform* and setting $S = 0.05$ and $M' = 9$.

For *Bandits*, we set B, the number of arms (or actions), to 5 and we use queries from D_{train}, whose fresh intent has shifted, to set the probability distribution over a query's fresh intent, used to define each bucket (see §3.2) using the fresh intent of each query after the shift to compute the probability distribution—as these parameters allow us to reach the best performance on D_{train}. To give some insight into the actions used, note that we end up with the following mean fresh intent values for each bucket: 0.18, 0.25, 0.31, 0.36, 0.45. Then, we derived the SERP for each action using the average SERP shown for each such fresh intent value by the search engine under study. We set p in order to reach a 14% precision level, which allows to reach the best performance on D_{train}.

4.2 Results

We use D_{train} for training our predictive model from §3.1, training the parameters of all exploration algorithms from §3.2, and use queries from D_{test} for the final evaluation (see §2.2). Results for all algorithms are shown in Table 3 (averaged over 10 runs of each simulation).

First, our *Bandit* algorithm allows us to reduce the cost from 37.66 to 35.48, without deteriorating the overall performance as the number of upgraded SERPs is much higher than the number of degraded ones (1,015,774 versus 711,947, that is +303,827). This limited performance is due to the fact that this algorithm spends too much time exploring each action as it takes a rather long time to determine which action is the best, while we do not have much time as we discussed in §3.1. Indeed, quickly finding which SERP is the best between the one associated with a fresh intent of 0.30 and 0.35 or even 0.41 is challenging when the median number of rounds is 81.

Second, *ExploreOnTop*, whether uniform or dynamic, allows us to greatly reduce both the cost and the median time delay (from 37.66 to 30.78 and 26,740 to 11,387, respectively, for the best performing algorithm) without deteriorating the overall performance as the number of upgraded SERPs is much higher than the number

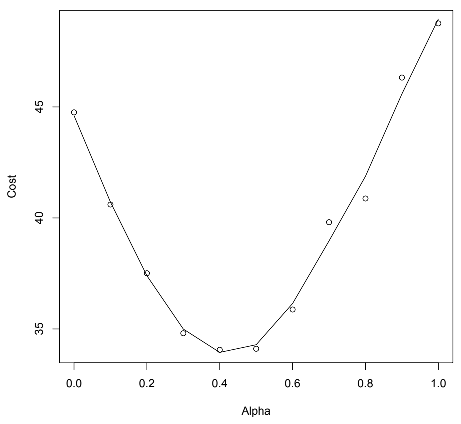


Figure 6: Cost versus α using *ExploreOnTop-Uniform* with p set to reach a 20% precision level and $M = 11$ on D_{train}.

of degraded ones (1,282,529 versus 445,197, that is +837,330 for the best performing algorithm). This means that even at $M = 11$, that is after exploring 11 issues of each selected query, user feedback is quite reliable and allows us to quickly correct fresh intent detection mistakes made by our fresh intent detector after an instant intent shift in most cases. Of course, increasing M makes user feedback even more reliable, but this comes at the cost of exploring much more, which is not worth it overall as the bias is already small at $M = 11$ (see Fig. 5). Dynamic exploration, even though it yields a slightly better performance, does not significantly outperform the uniform one; after only 8 or 9 issues, it is pretty difficult to know how good is our current estimate to decide whether to stop.

Even by setting $\hat{R}$ to 0.5, that is, by assuming that the first fresh result to be integrated at the top position of the SERP is always relevant to only half of the users with fresh intent, we obtain very good results. The cost and median time delay are very close to the ones obtained by setting $\hat{R}$ to 1, while the overall performance decreases but stays large at +325,700. Our algorithm is able to improve over our fresh intent detector even when the fresh vertical only provides fresh results that are not very relevant during exploration.

Depending on the quality of the search engine under study, we could investigate different ways of integrating fresh results on (top of) the SERP while exploring in order to get user feedback that is sufficiently informative. We could, for example, try to blend a few fresh results to increase the chance of at least one being relevant, or we could try to consider only very recent fresh documents when retrieving results from the fresh index. We leave this as future work.

Returning to our research questions RQ3 and RQ4, *Bandit* outperforms our fresh intent detector and our *ExploreOnTop* method outperforms both our fresh intent detector as well as *Bandit*. This interesting result is a consequence of the specifics of our setting, where the goal is to be faster than our fresh intent detector, and in which the opportunity to explore is limited. We thus need to take risks while exploring in order to be faster—as in our simple *ExploreOnTop* algorithm for which taking risks pays off.

5. RELATED WORK

We discuss related work on recency ranking and aggregated search.

Recency ranking. As discussed in §1, there are two main types of approach to recency ranking. One focuses on improving the ranking of generic web results by extending existing learning to rank algorithms to optimize for both freshness and relevance using temporal features. This is done either by training separate rankers for different classes of queries [5, 11] or by using a unified model that simultaneously optimizes freshness and relevance [8, 23].

An alternative approach to recency ranking, and the one that we use in this paper, avoids the complexity of the learning model of the approaches listed above as it uses a dedicated fresh vertical with its own documents, features and a ranking model that is specifically designed for dealing with recency ranking and that is independent from the web ranking model. This aggregated search-based approach is also more flexible than those approaches, where selecting a wrong ranker due to misclassification can substantially hurt the performance, as this approach allows one to integrate more or fewer fresh results depending on the *probability* of the query being fresh—thus dealing with cases when this value is uncertain [29]. In this context, we motivate the following algorithmic problem: how to quickly correct fresh intent detection mistakes made by our fresh intent detector after an instant intent shift.

Moon et al. [23] propose an algorithm for reranking the top portion of the SERP for recency sensitive queries using online exploration and real-time user feedback in order to keep track of the changes in the documents' relevance over time. In this paper, we also use online exploration and user feedback, but solve another problem related not to fresh ranking but to fresh intent detection.

Aggregated search. Diaz [9], Dong et al. [11], König et al. [20] propose different algorithms for identifying queries with a news intent that have been shown to be really accurate. The authors, however, do not discuss time delays in measuring the probability of the news intent, neither for new nor for recurring queries, including the queries whose newsworthiness has suddenly shifted. Therefore, they do not take into account the cost of the error in predicting this probability after such intent shifts. Our fresh intent detector also uses features similar to those in [9, 11, 20] (see §2.1). However, in this paper, we analyze time delays associated with these intent detections features observed in our setting, and the core challenge of our study is to learn to correct fresh intent detection mistakes and to rapidly recognize large changes in the fresh intent of a query when the detector is failing; a challenging problem that was not studied before. We solve this problem using online exploration, but as we cannot explore all queries for which a fresh intent shift might have happened, since we take the cost of exploration into account, we select the most appropriate queries for this treatment for which we then correct our fresh intent prediction using a linear model in order to properly integrate fresh results across the SERP.

Radinsky et al. [27] study how to model and predict behavioral dynamics on the Web such as query frequency or clicks using time series. In this paper, we use time series as features (see §3.1) and discuss how such techniques can be useful in our context, which differs from the ones studied in [27].

Jie et al. [19] study how to model the aggregated search task as a multi-armed bandit problem (MAB). The authors show how this approach can be used to deal with vertical selection and result merging in general, while we focus on the important problem of quickly correcting our predicted fresh intent for queries whose fresh intent has shifted. Here, their formulation as a MAB does not readily apply as we do not use slots to place fresh results on the SERP (see §3.2), and we propose a new formulation for our task thus complementing their study.

In sum, we contribute a new problem (rapid correction of fresh intent detection mistakes made by the state-of-the-art fresh intent detector after an instant intent shift), plus a new method to address the problem based on a semi-supervised predictive model trained from query logs and on online exploration phase.

6. CONCLUSION

In this paper, we have studied an aggregated search-based approach to recency ranking, where a (dedicated) *fresh vertical* is used and fresh results from this vertical are subsequently integrated into the search engine result page.

We presented a large scale analysis of fresh intent shifts upwards using query logs of Yandex. We observed that the state-of-the-art fresh intent detector often makes mistakes soon after a fresh intent shift occurred and may need a long time to properly detect the real fresh intent of a query with time delays of 9.5 hours on average. We also observed that the cost of the error in our prediction is important as indicated by the large percentage of queries for which fresh results could be inadequately integrated into the SERP due to the incorrectly predicted fresh intent of the query. Based on this, we motivated the following algorithmic problem: *how to quickly correct fresh intent detection mistakes made by this detector after an instant intent shift upwards*.

Then, we described a new method for solving this problem based on two phases: a phase in which we estimate the probability for the fresh intent of a query to have shifted considerably after we started to detect a small increase in its predicted fresh intent using a semi-supervised learning approach and a subsequent phase in which we perform an exploratory feedback step online.

Using query logs of Yandex, we demonstrated that our method, when using a *multi-armed bandits* approach to online exploration for the second phase, allows us to reduce the cost of fresh intent detection mistakes made by the state-of-the-art detector after an instant intent shift, without deteriorating the overall performance. We also demonstrated that our method, when using a new *explore on top* approach to online exploration, where we briefly show a fresh result on top of the SERP and then exploit user feedback to correct the predicted fresh intent, allows us to significantly improve the speed and quality of the detection of fresh intent shifts and outperforms the *multi-armed bandits* approach.

We have shown that a system using the aggregated search-based approach to recency ranking, which uses a fresh intent detector, can benefit from this work and from our proposals to significantly improve the speed and the quality of the detection of intent shifts without deteriorating overall retrieval performance.

Our semi-supervised learning approach opens the opportunity to test new real-time features, which would not be very strong in our fresh intent detector, but which could yield good results in our predictive model from §3.1 in order to further reduce the cost and time delays connected to fresh intent shift detection—with less degradation. Our exploration algorithms could also be improved by investigating ways of integrating fresh results on (top of) the SERP for our *explore on top* approach as discussed in §4.2, or by investigating ways to make *multi-armed bandits* converge faster.

Acknowledgements. We thank Fedor Romanenko for originally proposing the idea for an algorithm for detecting shifts in fresh intent as well as the whole Yandex's freshness team for interesting discussions and support.

This research was supported by the European Community's 7th Framework Programme (FP7/2007-2013) under grant agreements nr 288024 (LiMoSINe) and 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, the Center for Creation, Content and Technology (CCCT), 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.

7. REFERENCES

[1] J. Arguello, J. Callan, and F. Diaz. Classification-based resource selection. In *CIKM '09*, pages 1277–1286. ACM, 2009.

[2] J. Arguello, F. Diaz, J. Callan, and J.-F. Crespo. Sources of evidence for vertical selection. In *SIGIR '09*, pages 315–322. ACM, 2009.

[3] J. Arguello, F. Diaz, and J. Callan. Learning to aggregate vertical results into web search results. In *CIKM '11*, pages 201–210. ACM, 2011.

[4] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *FoCS '95*, pages 322–331. IEEE, 1995.

[5] J. Bian, T.-Y. Liu, T. Qin, and H. Zha. Ranking with query-dependent loss for web search. In *WSDM '10*, pages 141–150. ACM, 2010.

[6] N. Cesa-Bianchi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[7] O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *NIPS '11*, pages 2249–2257, 2011.

[8] N. Dai, M. Shokouhi, and B. D. Davison. Learning to rank for freshness and relevance. In *SIGIR '11*, pages 95–104. ACM, 2011.

[9] F. Diaz. Integration of news content into web results. In *WSDM '09*, pages 182–191. ACM, 2009.

[10] F. Diaz and J. Arguello. Adaptation of offline vertical selection predictions in the presence of user feedback. In *SIGIR '09*, pages 323–330. ACM, 2009.

[11] A. Dong, Y. Chang, Z. Zheng, G. Mishne, J. Bai, R. Zhang, K. Buchner, C. Liao, and F. Diaz. Towards recency ranking in web search. In *WSDM '10*, pages 11–20. ACM, 2010.

[12] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: improving recency ranking using twitter data. In *WWW '10*, pages 331–340. ACM, 2010.

[13] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[14] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*, pages 124–131. ACM, 2009.

[15] T. Hastie, R. Tibshirani, and J. J. H. Friedman. *The elements of statistical learning*, volume 1. Springer New York, 2001.

[16] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *ECIR 2011*, 2011.

[17] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM 2011*, October 2011 2011.

[18] 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.

[19] L. Jie, S. Lamkhede, R. Sapra, E. Hsu, H. Song, and Y. Chang. A unified search federation system based on online user feedback. In *KDD '13*, pages 1195–1203. ACM, 2013.

[20] A. C. König, M. Gamon, and Q. Wu. Click-through prediction for news queries. In *SIGIR '09*, pages 347–354. ACM, 2009.

[21] M. Lalmas. Aggregated search. In *Advanced Topics in Information Retrieval*, pages 109–123. Springer, 2011.

[22] D. Lefortier, P. Serdyukov, F. Romanenko, and M. de Rijke. Blending vertical and web results: A case study using video intent. In *ECIR 2014*, 2014.

[23] T. Moon, W. Chu, L. Li, Z. Zheng, and Y. Chang. An online learning framework for refining recency search results with user click feedback. *ACM Trans. on Information Systems*, 30(4):20, 2012.

[24] H. Nottelmann and N. Fuhr. From retrieval status values to probabilities of relevance for advanced ir applications. *Information retrieval*, 6(3-4):363–388, 2003.

[25] A. Ponnuswami, K. Pattabiraman, Q. Wu, R. Gilad-Bachrach, and T. Kanungo. On composition of a federated web search result page. In *WSDM '11*, pages 715–724. ACM, 2011.

[26] A. K. Ponnuswami, K. Pattabiraman, D. Brand, and T. Kanungo. Model characterization curves for federated search using click-logs: predicting user engagement metrics for the span of feasible operating points. In *WWW '11*, pages 67–76. ACM, 2011.

[27] K. Radinsky, K. M. Svore, S. T. Dumais, M. Shokouhi, J. Teevan, A. Bocharov, and E. Horvitz. Behavioral dynamics on the web: Learning, modeling, and prediction. *ACM Trans. on Information Systems*, 31(3):16, 2013.

[28] M. Sloan and J. Wang. Iterative expectation for multi period information retrieval. *arXiv preprint arXiv:1303.5250*, 2013.

[29] A. Styskin, F. Romanenko, F. Vorobyev, and P. Serdyukov. Recency ranking by diversification of result set. In *CIKM '11*, pages 1949–1952. ACM, 2011.