Journal of Web Engineering, Vol. 14, No. 5&6 (2015) 386–413

FROM TMR TO TURTLE: PREDICTING RESULT RELEVANCE FROM MOUSE CURSOR INTERACTIONS IN WEB SEARCH

MAXIMILIAN SPEICHER[1,a], SEBASTIAN NUCK[2], LARS WESEMANN[3], ANDREAS BOTH[3] and MARTIN GAEDKE[1]

[1] *Department of Computer Science, Technische Universität Chemnitz*
09111 Chemnitz, Germany
{*maximilian.speicher@s2013|martin.gaedke@cs*}*.tu-chemnitz.de*

[2] *Agile Knowledge Engineering and Semantic Web Group, Leipzig University*
04109 Leipzig, Germany
nuck@infai.org

[3] *Research & Development, Unister GmbH*
04109 Leipzig, Germany
{*lars.wesemann|andreas.both*}*@unister.de*

The prime aspect of quality for search-driven web applications is to provide users with the best possible results for a given query. Thus, it is necessary to predict the relevance of results *a priori*. Current solutions mostly engage clicks on results for respective predictions, but research has shown that it is highly beneficial to also consider additional features of user interaction. Nowadays, such interactions are produced in steadily growing amounts by internet users. Processing these amounts calls for streaming-based approaches and incrementally updatable relevance models. We present *StreamMyRelevance!*—a novel streaming-based system for ensuring quality of ranking in search engines. Our approach provides a complete pipeline from collecting interactions in real-time to processing them incrementally on the server side. We conducted a large-scale evaluation with real-world data from the hotel search domain. Results show that our system yields predictions as good as those of competing state-of-the-art systems, but by design of the underlying framework at higher efficiency, robustness, and scalability.

Additionally, our system has been transferred into a real-world industry context. A modified solution called *Turtle* has been integrated into a new search engine for general web search. To obtain high-quality judgments for learning relevance models, it has been augmented with a novel crowdsourcing tool.

Keywords: Case Study, Crowdsourcing, Industry, Interaction Tracking, Learning to Rank, Real-Time, Relevance Prediction, Streaming

1 Introduction

In a World Wide Web that has grown to a size of at least 4.48 billion pages,[b] search engines are among the most important and popular web applications, as they make enormous amounts of content reachable. The prime aspects that determine the success of a search engine are delivering relevant results and displaying them using the best possible ranking. If one of

[a] The contents of this article were developed while Maximilian Speicher stayed at Unister GmbH as an industrial PhD student.

[b] `http://worldwidewebsize.com/` (Oct 02, 2014).

these is not met—e.g., the most relevant result is not present at least on the first page—the search engine is *de facto* useless for the user. However, before delivering results, we can only *guess* their relevance for the user based on a number of factors, such as previous searches. A common approach is to make use of generative *click models*, which try to predict the perceived relevance of results based on the clicks they have received and certain assumptions about user behavior. To give just one example, the *Cascade Model* [1] assumes that users examine results from top to bottom and do not pay attention to results below their first click. Yet, clicks are not a perfect indicator for relevance. For example, some users might tend to open a number of results without paying closer attention at first. Also, search engines more and more try to answer certain queries directly on the *search engine results page* (SERP), as facilitated by, e.g., the *DuckDuckGo Instant Answer API*.[c] In such cases, the user does not have to click at all anymore. Thus, additional information that complement click data should be taken into account for predicting relevance, e.g., in terms of dwell times on landing pages [2] or other client-side user behavior (e.g., [2, 3, 4]). Previous research has shown the value of such page-level interactions [5, 3, 6]. Also, generative [7] as well as discriminative [6] approaches to relevance prediction exist that engage user behavior other than clicks only.

However, with growing numbers of users and today's asynchronous client-side technologies, it is possible to collect vast amounts of user interactions. In particular, this applies if we consider interactions other than clicks. Moreover, in today's IT industry, a short time-to-market is gaining importance. That is, to ensure user satisfaction, search engine providers need to analyze collected information as fast as possible and feed their findings directly back into the ranking process. The most efficient way to do so is to build on *streams* of data and process them in *real-time*. This calls for the use of novel systems for data stream mining, such as the distributed real-time computation system *Storm*,[d] which are currently gaining popularity in research and industry. These systems can help to cope with the seemingly endless streams of data produced by Internet users. Yet, none of the approaches for relevance prediction mentioned above leverages data stream mining to process collected tracking data.

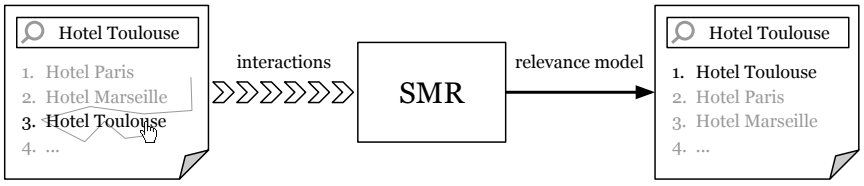

Fig. 1. The intention behind StreamMyRelevance!—from collecting a stream of user interactions to reordering search results based on relevance models.

We present ***StreamMyRelevance!* (SMR)**, which is a novel streaming-based system for ensuring ranking quality in search engines. Our system caters for the whole process from tracking interactions—in terms of *mouseenter, -click, -move and -leave* actions on search results—and relevance judgments to learning incremental *relevance models*. That is, models that predict the relevance of a search result (for a given query) based on certain features of user interaction—like *hover duration and cursor trail/speed*—that are derived from the tracked

[c] `https://duckduckgo.com/api` (Oct 14, 2014).
[d] `http://storm.apache.org/` (Oct 14, 2014).

mouse actions. The latter can be used to directly feed predictions back into the ranking process of the search engine, e.g., as a weighted factor in a learning-to-rank function (cf. Fig. 1). SMR is based on Storm and leverages tracking and data processing functionalities provided by *TellMyRelevance!* (TMR)—a pipeline that has proven its effectiveness in predicting search result relevance [6]. Yet, TMR is a batch-oriented approach that does not provide means for incrementally learning relevance models on a streaming basis. Thus, SMR wraps the borrowed functionalities into a new system that is able to handle real-time streams. Our system has the following three advantages over existing approaches:

1. It considers *interactions beyond clicks* for predicting relevance.
2. It collects and processes these interactions in terms of a *stream*.
3. It provides *incremental relevance models* that do not require reprocessing of previously processed data.

Based on this, the core hypothesis investigated in this article reads as follows: *SMR is able to achieve the same relevance prediction quality as TMR at better efficiency, robustness and scalability.*

We have evaluated SMR in terms of its feasibility and quality of relevance predictions. For this, ∼23 GB of real-world interaction tracking data from two hotel booking portals were available. In this context, completed hotel booking processes were treated as indicators of result relevance for learning models. A comparison to TMR has been performed, which due to its batch-oriented design has look-ahead capabilities and thus more information available. Still, our results show that SMR's prediction quality is not significantly worse compared to TMR. Moreover, our system in parts compares favorably with predictions of the Bayesian Browsing Model (BBM) [8], a *state-of-the-art* generative click model successfully applied in industry. Furthermore, reviews of efficiency, robustness and scalability show that SMR compares favorably with the competing approaches in these respects.

Finally, we have transferred our novel approach into a real-world industry context. That is, a specifically tailored version of our pipeline—named ***Turtle***—has been integrated into the architecture of a new search engine developed by the R&D department of the cooperating company [9]. In this context, we build on a hybrid solution between TMR and SMR. Also, unlike in the evaluation of SMR, we made use of an internal *crowdsourcing* tool for obtaining relevance judgments.

In the following section, we describe important concepts our work is based on, before giving an overview of related work. Section 3 explains the design and architecture of SMR as well as the component architecture and integration of Turtle. This is followed by an evaluation of effectiveness, efficiency, robustness and scalability of SMR and competing approaches in Section 4. Limitations and potential future work are addressed in Section 5 before giving concluding remarks in Section 6.

2 Background and Related Work

The following gives background information on the underlying concepts of Storm [10], which are important for understanding the architecture of SMR.

The logic of a Storm application is represented as a graph consisting of **spouts** and **bolts** that are connected by *streams*, i.e., unbounded sequences of data tuples. This concept is called

a **topology**. On the one hand, spouts act as sources of streams by reading from external data sources (e.g., a DB) and emitting tuples into the topology. On the other hand, bolts are the core processing units of a topology. They receive tuples, process the contained data and emit results as a new stream. Spouts and bolts can have multiple outgoing streams, which provides the possibility of separating tuples within bolts and emitting them using different streams.

The direct competitor to Storm is Yahoo!'s S4.[e] It as well provides distributed stream computing functionality, but its underlying concepts and configuration are more complex.[f] As described in [11], benchmarks have shown that S4 is almost 10 times slower than Storm.

Our research is related to a variety of existing work in the fields of *relevance prediction* and *data stream mining*. An overview will be given in the following.

Concerning the relevance of search results, it is necessary to rely on human relevance judgments—i.e., asking the user to explicitly rate the relevance of a result—for the best possible predictions. However, since such data are usually not available in large numbers, different solutions are required. Joachims [12] proposes to use *clickthrough data* instead of human relevance judgments. Based on the *cascade hypothesis* [1, 8], i.e., the user examines results top-down and neglects results below the first click, it is possible to infer relative relevances. That is, the clicked result is more relevant than the non-clicked results at higher positions. Using such relative relevances, Joachims engages clickthrough data as training data for learning retrieval functions with a support vector machine approach [12]. In contrast to the above, models like the *Dependent Click Model* [13] assume that more than one result can receive clicks. That is, results below a clicked position might be examined and thus also clicked if they are relevant.

The *Dynamic Bayesian Network Click Model* (DBN) described in [14] generalizes the *Cascade Model* [1] by aiming at relevance predictions that are not influenced by position bias. To achieve this, the authors (besides the perceived relevance of a search result) also consider users' satisfaction with the website linked by the clicked result.

Generally, click models are based on the *examination hypothesis*, which states that only relevant search results that have been examined are clicked [8]. Yet, not all of these models follow the cascade hypothesis. All of the above described are *generative* click models that try to provide an alternative to explicit human judgments by *predicting the relevance of search results* based on click logs. The main differences to our systems—TMR and SMR—are that we aim at predicting relevance using a *discriminative* approach also taking into account *interactions other than clicks*. Moreover, unlike SMR, the above click models are not designed for efficient processing of *massive data streams* or *incremental updates*.

The *Bayesian Browsing Model* (BBM) [8] is based on the *User Browsing Model* (UBM) [15], which assumes that the probability of examination depends on the position of the last click and the distance to the current result [8]. Contrary to UBM, BBM aims at scalability to petabyte-scale data and incremental updates. The authors compute "relevance posterior[s] in closed form after a single pass over the log data" [8]. This enables incremental learning of the click model while making iterations unnecessary. Still, contrary to SMR, BBM is again a *generative* model that does not leverage the advantages of additional interaction data.

[e] `http://incubator.apache.org/s4/` (Sep 28, 2013).
[f] `http://demeter.inf.ed.ac.uk/cross/docs/s4vStorm.pdf` (Jan 06, 2014).

Concerning user interactions other than clicks, in [5], Huang has found that these are a valuable source of information for relevance prediction. Following, Huang et al. [3] investigate the correlations between human relevance judgments and mouse features such as hover time and unclicked hovers, among others. They find positive correlations (e.g., r=0.46 between human judgments and the hover rate of clicked results) and conclude that these can be used for inferring search result relevance. Also, part of our system is based on a scalable approach for collecting client-side interactions described by the authors [3].

In [2], Guo and Agichtein present their *Post-Click Behavior Model*. They incorporate interactions like cursor or scrolling speed on a landing page into determining its relevance, i.e., interactions that happen *post-click*. This is also partly related to DBN [14], where the relevance of the landing page is modeled separately from the perceived relevance of the result. While this approach is promising for inferring the actual usefulness of a landing page, it would be difficult to realize since search engines would need access to landing page interactions through, e.g., a browser plug-in or tracking scripts.

Making use of scrolling and hover interactions, Huang et al. [7] extend the Dynamic Bayesian Network Click Model described earlier to leverage information beyond click logs. Their results show that this improves the performance in terms of predicting future clicks compared to the baseline model. While this *generative* approach involves interactions other than clicks, in contrast to SMR, it does not specifically aim at incremental learning or efficient processing of massive data streams.

TellMyRelevance! TMR is a system described by Speicher et al. [6]. Parts of SMR are based on this work, particularly in terms of client-side interaction tracking, preprocessing of raw data and computation of interaction features. Like SMR, TMR is a *discriminative* approach to relevance prediction, but in contrast is a batch-oriented system. That is, raw tracking data have to be fetched from the key-value store at predefined intervals and it is not possible to learn incremental classifiers. Instead, we need to completely reprocess *all* mouse features and relevance judgments if we want to update an already existing model. Assume we want to obtain an up-to-date model once a day. Then, at some point in time, it would take longer than 24 hours to (re-)process all data that is required for the update, or the system would have to scale accordingly, e.g., by adding faster hardware. Thus, for the sake of feasibility it is necessary to have a solution that processes data once and only once. That is, a *streaming*-based pipeline that works on a per–search session basis and learns a model *incrementally* that is automatically fed back into the ranking process of the corresponding search engine.

3 SMR: Streaming Interaction Data for Learning Relevance Models

Scenario and Requirements A large e-commerce company is developing a new semantic search engine for travel search. For this, they are crawling a huge amount of relevant pages that are fed into their own search index, i.e., they do not build on external APIs. Delivered results are ordered according to a complex ranking function that comprises 29 query-independent (e.g., length of URL) and query-dependent (similarity between query and features of a web-page) ranking factors. One of these factors is determined by a relevance model that analyzes clicks from past search sessions and is updated once a week. However, from current research

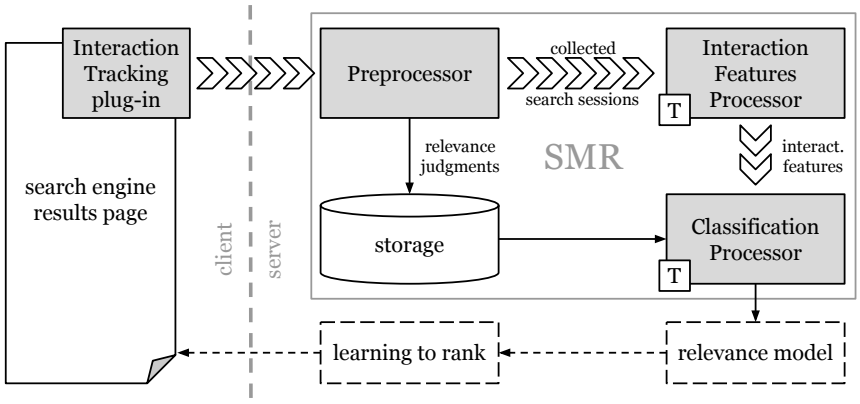

Fig. 2. The main components and process flow of SMR (Streams are visualized by sequences of chevrons; Storm topologies are annotated using a "T").

papers the head of R&D knows of the additional value when also leveraging *additional mouse interactions*. These are collected in *real-time* for different tracking purposes anyway. Thus, the company intends to build on a new approach also considering such interactions beyond clicks. Moreover, the head of R&D demands to have an *up-to-date relevance model* at any given point in time to be able to provide the best possible searching experience even in rapidly changing environments.

From the above scenario, we can derive the following requirements that have to be met by a corresponding system satisfying the company's demands:

(R1) Interactions beyond clicks The system pays attention to user behavior other than clicks (e.g., cursor trail, hovers etc.) for predicting result relevance.

(R2) Stream processing The system processes all tracking and intermediate data in (near) real-time in terms of streams.

(R3) Incremental model The delivered relevance model can be updated in an incremental manner. That is, only the current chunk of information must be added to the previous version of the model for an update. No reprocessing of previously processed data is necessary.

We present *StreamMyRelevance!* (SMR)—a system meeting these requirements, which is organized as a streaming-based process and described in the following sections. Its aim is to enable processing of big data streams while leveraging the advantages of user interaction data for the prediction of search result relevance. This supports more optimal ranking of results, which is a major quality aspect of search-driven web applications.

The system comprises four main components as illustrated in Fig. 2: The **Client-Side Interaction Tracking** component in terms of a jQuery plug-in; The **Preprocessor** for reading and preprocessing streams of tracking data and relevance judgments; The **Interaction Features Processor** for calculating interaction features from tracking data; The **Classification Processor** for incrementally training a relevance model using the previously computed features and collected relevance judgments.

Our Storm-based system has been specifically designed with an incremental approach in mind. The four steps above can be regarded as a sequence of independent processes. That is, the results of each step as well as the resulting relevance models are persisted (temporarily). As a result, in case of a crash within the system, SMR can resume its work at the step prior to the incident without starting over from the very beginning.

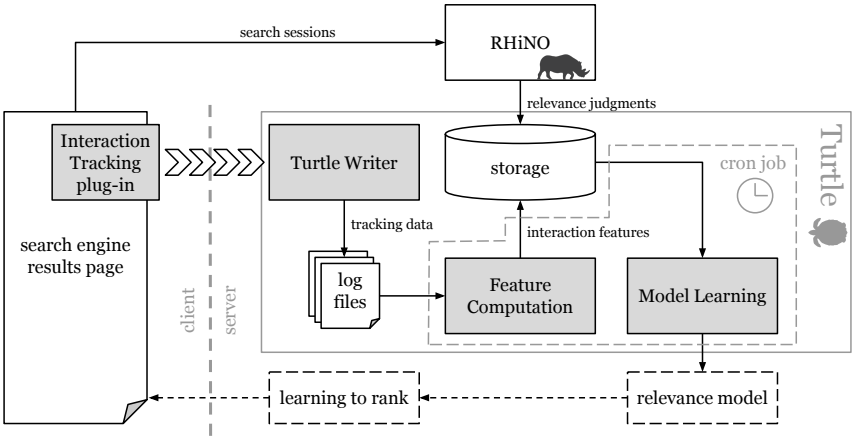

Fig. 3. Architecture of Turtle, a hybrid TMR/SMR solution in a real-world industry context.

Furthermore, we have integrated our system into a real-world search engine currently developed by the R&D department of the cooperating company (Fig. 3). The search engine is based on a service-oriented architecture that combines means for full-text, geo-spatial and semantic search [9]. It features different search modes, including hotel and the commonly known web search. The integrated system is a hybrid approach named *Turtle* that engages parts of TMR [6] and SMR and has been incorporated into the search engine's process flow in a real industry context.

For ordering delivered results, the novel search engine currently engages a ranking function that includes a total of 29 weighted features, such as the length of the URL or the number of `div` tags contained in a webpage. These features are computed for each result for a given query requested by the user. The overall score of a result according to the ranking function and assigned weightings then determines its rank on the SERP. Our aim is to incorporate *Turtle relevance* as an additional feature into the search engine's ranking function. In the following, the implementation of Turtle will be described component-wise along the lines of the system description of SMR.

3.1 Client-Side Interaction Tracking

3.1.1 Component Description

For client-side interaction tracking, SMR builds upon a "minimally invasive jQuery plug-in" [6] that is provided by TMR. This plug-in tracks `mouseenter`, `mousepause`, `mousestart`, `mouseleave`, `click` and `clickthrough` events—as also specified in Fig. 5 below—that happen within the bounds of a search result on a SERP *(R1)* [6]. Each mouse event is extended with the search query, an anonymized user ID and the ID of the corresponding result [6].

The resulting data packets are then sent to a specified key-value store at suitable intervals (Fig. 2) [6]. For integration, the developer has to specify jQuery selectors corresponding to the observed SERP for (a) the HTML container element holding all results, (b) a single search result, (c) an element within a result holding the result ID and (d) links to landing pages, according to the following example:

```
var sid = (new Date()).getTime();
$.SMR({
  /* the HTML <div> element with ID 'wrapper' contains all results */
  resultItemContainer: 'div#wrapper',
  /* each result is held by a <div> element with CSS class 'result' */
  resultItemSelector: 'div.result',
  /* the link with CSS class 'landingPage' holds the ID of the respective result */
  resultIDSelector: 'a.landingPage',
  /* the link with CSS class 'landingPage' leads to the actual result URL */
  linkSelector: 'a.landingPage',
  /* function extracting the result ID from the element identified by resultIDSelector */
  getID: function($element) {
    return $element.attr('data-resultId');
  },
  query: 'Hello, World!', //the current query
  pageNo: 1, //the current page number
  sessionID: sid //a unique session ID
});
```

The second function provided by the plug-in is intended for recording human relevance judgments, often also referred to as *conversions*, which are crucial for learning relevance models. It is realized as a JavaScript method that can be called from anywhere, e.g., upon clicking an upvote button next to a search result [6]. This method has to be provided with the value of the judgment (e.g., -1 for a downvote and $+1$ for an upvote) as well as the corresponding search query, session ID and user ID by the developer.

3.1.2 Algorithm

The following pseudocode summarizes the precise functionality of the client-side interaction tracking component:

```
module interactionTracking {
  metadata ← (searchSessionID, userID, query, listOfResults, timestamp)
  add metadata to buffer

  for each result r on SERP {
    for each event e in (mouseenter, mousepause, mousestart, mouseleave) {
      /* add event listener */
      when e occurs on r {
        data ← (searchSessionID, resultID, eventObject)
        add data to buffer
      }
    }
  }

  repeat {
    if buffer size ≥ threshold {
      dataPacket ← compress(buffer)
      send dataPacket to preprocessor
      empty buffer
    }
  }
}
```

```
module relevanceJudgments {
  function registerJudgment(value) {
    judgment ← (searchSessionID, userID, resultID, value)
    dataPacket ← compress(judgment)
    send dataPacket to preprocessor
  }
}
```

3.1.3 Industrial Integration

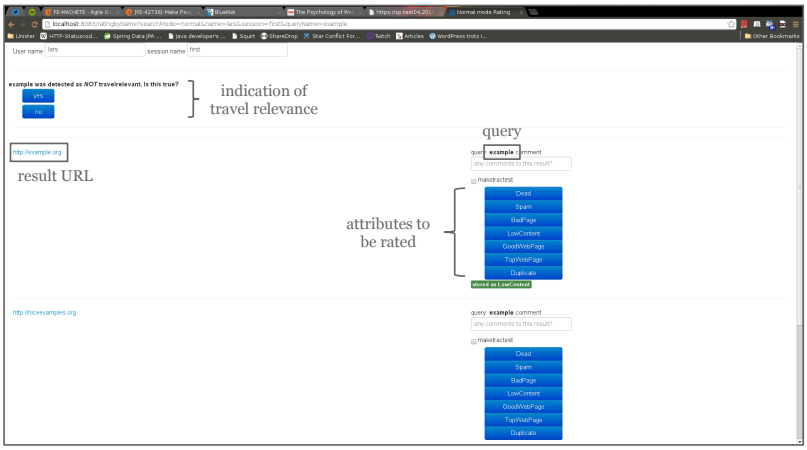

Fig. 4. The RHiNO tool, as shown to crowd workers, for the query "example" with results `http://example.org` and `http://niceexamples.com`.

In the context of Turtle, the means for client-side interaction tracking have been integrated as described above. All interactions are stored in dedicated log files by the *Turtle Writer* (Fig. 3), whereas one log file corresponds to one day. Moreover, we obtain relevance judgments through *RHiNO* (Quality *R*ating Tool for *H*otel and *NO*rmal search), which is a company-internal crowdsourcing tool specifically developed for rating different aspects of search results. As input, the tool requires search sessions in XML format. A search session comprises a query and the corresponding results the search engine displays for that query. In particular, it is possible to post all search sessions triggered by real users to RHiNO to obtain judgments for results users actually see. Yet, it is also possible to have manually compiled lists of search sessions evaluated, e.g., if trending queries are expected in the future.

The posted search sessions are then randomly displayed to the crowd workers, one search session at a time (Fig. 4). It is explicitly possible that a search session is reviewed multiple times by different evaluators. For each result contained in a session, the crowd worker has to provide yes/no feedback on seven attributes:

bad Is the result not relevant w.r.t. the given query?
dead Does the result lead to a dead page?
duplicate Is the result a duplicate of another result in the search session?
good Is the result relevant w.r.t. the given query?
low content Does the result lead to a page with very little content?

spam Does the result lead to a spam page?
top Is the result the best possible result w.r.t. the given query?

The ratings of these attributes are then stored to a database per query–result pair (yes = 1, no = 0). That is, notwithstanding the algorithm given above, RHiNO bypasses the processing of relevance judgment by SMR's server-side components and instead instantly stores them along with the necessary meta-information. If ratings are already present for a query–result pair, they are summed up. The total number of ratings is stored as well for normalization purposes.

3.2 Preprocessor

3.2.1 Component Description

After having been recorded using the above jQuery plug-in, all interaction data is received by SMR as a *stream of individual events (R2)* for preprocessing (Fig. 2). Additionally, information about a corresponding search session[g] is transferred when a user enters a SERP. These contain an anonymous user ID, the current search query and the ordered list of all results, among others [6]. Every event received by SMR is subsequently associated with its respective search session. This concept is referred to as a *collected search session*.

It is logically not possible to process events from search sessions that have not ended yet. Thus, all events are passed on in the SMR pipeline on a per–search session basis *(R2)*. Since with current web browser implementations it is unreliable to fire client-side `unload` events on a SERP, this is realized using a configurable time-out on the server side. For example, if no events related to a given session have been received for 2 minutes, it is considered finished and the collected search session is passed on for interaction feature computation (Fig. 2).

Moreover, the preprocessing component receives human relevance judgments that are required for learning actual models. These judgments are checked for validity, i.e., whether a corresponding search session exists during which the judgment happened. The latter is not the case if a judgment is triggered by a user who did not perform a search beforehand, e.g., because they received a link to a result from a friend. Relevance judgments are persisted at this point for later use by the Classification Processor (Fig. 2). Finally, for later filtering purposes, each valid judgment is associated with the list of queries triggered by the corresponding user ID.

3.2.2 Algorithm

The following pseudocode summarizes the precise functionality of the prepocessor:

```
module preprocessor {
  receive dataPacket
  data ← decompress(dataPacket)

  for each object in data {
    ID ← object.searchSessionID

    switch object {
    case search session metadata:
      create collectedSearchSession[ID]
```

[g]For our purposes, a search session starts when entering and ends when leaving a SERP. For example, a reload triggers a new session, even for the same user and query.

```
      add metadata to collectedSearchSession[ID]
      save collectedSearchSession[ID]
    case mouse event:
      fetch collectedSearchSession[ID]
      add event to collectedSearchSession[ID]
      save collectedSearchSession[ID]
    case judgment:
      if (collectedSearchSession[ID] exists) {
        queries ← all collectedSearchSession[•].query where
          collectedSearchSession[•].userID = judgment.userID
        add queries to judgment
        save judgment
      }
    }
  }

  repeat {
    for all collectedSearchSession[ID] {
      if collectedSearchSession[ID] not modified for longer than threshold {
        send collectedSearchSession[ID] to interactionFeaturesProcessor
        delete collectedSearchSession[ID]
      }
    }
  }
}
```

3.2.3 Industrial Integration

The integration of the preprocessing component into the real-world industry context has been realized in analogy to the above. Yet, at this point in time, the architecture of the search engine is not streaming-oriented and thus not based on Storm [9]. Hence, it is reasonable to neglect the Storm functionality of the preprocessor for now—according to the YAGNI principle ("You aren't gonna need it")[h]—and instead build on a batch-wise approach, as is also done by TMR [6].

For updating the scores of the search engine's ranking function, Turtle must provide an up-to-date relevance model on a weekly basis. For this, a *cron job* fetches the raw interaction tracking logs created during the past week and has them prepared by the preprocessor included in Turtle (contained in the component *Feature Computation* in Fig. 3).

3.3 Interaction Features Processor

3.3.1 Component Description

The Interaction Features Processor is realized as a separate topology within our Storm-based system (Fig. 2). It receives *collected search sessions* from the preprocessor that are emitted as a stream by a dedicated spout *(R2)*. To ensure that all interaction events associated with a search session are ordered logically, invalid sequences of events are filtered out based on a finite state machine (Fig. 5). This prevents the computation of faulty interaction feature values. An invalid sequence would be, e.g., if a `mouseleave` happens before a `mouseenter` event on the same search result. Typical causes for such a case can be faulty time stamps or latency while transferring data from client to server. Since at the moment we specifically focus on *mouse* interactions, search sessions that have been recorded on touch devices are eliminated as well.

[h] Cf. `http://www.wikiwand.com/en/You_aren't_gonna_need_it` (Oct 22, 2014).

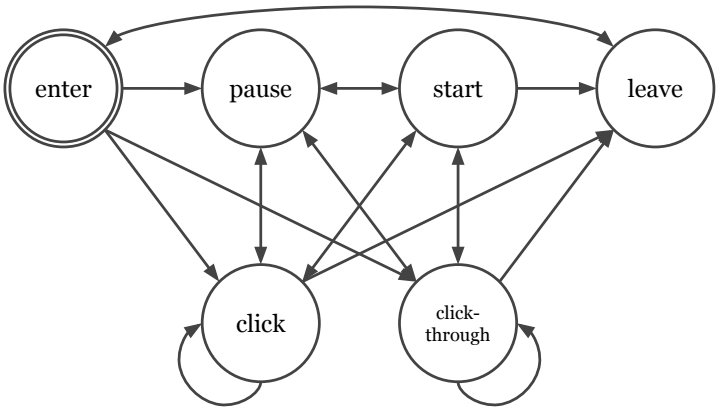

Fig. 5. The finite state machine defining the possible transitions between mouse events captured on the client side.

Subsequently, the values of the actual interaction features are calculated per query–result pair. For example, the value of the *arrival time* is determined by subtracting the time stamp of the first `mouseenter` event on a result from the time stamp of the page load (which is available as meta information about the associated search session). The features we are considering have been compiled from a variety of existing research and are in accordance with [6] *(R1)*:

Hover time* The amount of time the mouse cursor spent within the bounds of the result [3, 4].

Cursor movement time* The amount of time during which the cursor actually moved within the bounds of the result, i.e., *hover time* minus the time the cursor stood still [6].

Arrival time The amount of time between entering the SERP and the cursor entering the result for the first time [3, 4].

Clickthroughs* The amount of clicks on hyperlinks leading to the result's *landing page*, i.e., the website the user ultimately wants to visit [3].

Clicks The amount of clicks on hyperlinks (within the bounds of the result) not leading to the landing page, e.g., links showing additional info boxes [6].

Hovers* The amount of hovers over the result [3, 4].

Unclicked hovers The amount of hovers not resulting in a `clickthrough` event [3].

Maximum hover time* The time duration of the longest single hover action over the result [3].

Cursor trail The amount of pixels the cursor traveled within the bounds of the result [2, 4].

Cursor speed *Cursor trail* divided by *cursor movement time* [2].

Position The position of the result within the SERP, also considering the SERP's page number; i.e., on the second SERP for a given query, the first result has position 11, assuming ten results are shown per SERP [3, 2].

* indicate the interactions features that showed the highest correlations with result relevance in an interaction log analysis that is described later in this article. The features are moreover averaged over the number of hovers, if possible. This applies to *clicks, clickthroughs,*

cursor movement time, cursor trail, hover time and *unclicked hovers* [6]. Finally, the computed values are persisted, which is important for later normalization purposes and actual use of SMR's relevance models (see below). In case feature values are already present for a query–result pair, they are automatically updated by adding the new values and taking the average over all values.

Within this topology, emitting a stream of collected search sessions is realized using a *spout (R2)*. Contrary, checking event sequence validity, the actual computation of feature values and updating values of already existing query–result pairs are realized through *bolts*.

The raw search sessions and associated events are not necessarily lost after they have been used for computing interaction features. Rather, SMR provides the option to persist all processed data. In this way, it is possible to batch-wise train a new model from parts of old data (e.g., after removing outdated information) before continuing to incrementally update this new model using real-time interactions and judgments.

3.3.2 Algorithm

The following pseudocode summarizes the precise functionality of the interaction features processor:

```
module interactionFeaturesProcessor {
  receive collectedSearchSession[ID]

  for each event in collectedSearchSession[ID] {
    q ← collectedSearchSession[ID].query
    r ← event.resultID

    /* e.g., filter out mouseleave events with no corresponding mouseenter */
    check event for validity according to Fig. 5

    /* calculate interaction features, e.g. */
    if event is mouseleave {
      featureValues[q,r].hoverTime += (event.timestamp - lastMouseenter.timestamp)
    }

    ...
  }

  /* for simplicity, q := query, r := resultID */
  for all featureValues[q,r] {
    for each f in (clicks, clickthroughs, cursorMovementTime, cursorTrail, hoverTime,
        unclickedHovers) {
      featureValues[q,r].f /= featureValues[q,r].hovers
    }

    if persistedFeatures[q,r] exists {
      persistedFeatures[q,r] ← (persistedFeatures[q,r] + featureValues[q,r]) /
        ++persistedFeatures[q,r].count)
    } else {
      persistedFeatures[q,r] ← featureValues[q,r]
      persistedFeatures[q,r].count ← 1
      save persistedFeatures[q,r]
    }

    send persistedFeatures[q,r] to classificationProcessor
  }
}
```

3.3.3 Industrial Integration

The integration of the interaction features processing component into the real-world industry context has been realized in analogy to the above, i.e., Turtle computes interaction features per query–result pair and permanently stores them to a database (Fig. 3, *Feature Computation*). Yet, at this point in time, the architecture of the search engine is not streaming-oriented and thus not based on Storm [9]. Hence, it is reasonable to neglect the Storm functionality of the interaction features processor for now—according to the YAGNI principle—and instead build on a batch-wise approach, as is also done by TMR [6].

3.4 Classification Processor

3.4.1 Component Description

The Classification Processor is as well realized as a separate topology within our system (Fig. 2). It receives the previously calculated *interaction features* (one set per query–result pair) in terms of a stream that is emitted into the Storm cluster by a dedicated spout *(R2)*. Using the lists of queries associated to judgments during preprocessing, we filter out sets of interaction feature values that are not associated with a user who triggered at least one relevance judgment. This helps to ensure a good quality of our training data.

Moreover, relevance models provided by SMR highly depend on the layout of a SERP [6]. Thus, normalization of feature values is necessary to guarantee comparability between models related to different SERP layouts [6]. This happens in terms of dividing feature values by the maximum value of the respective feature across all results for the given query. Since interaction feature values arrive as a stream, maximum values change over time and have to be constantly updated. Hence, they become more precise the longer the system runs. This is a major difference compared to TMR, which—due to its batch-oriented nature—has look-ahead capabilities and knows exact maximum values from the start.

In the next step, we derive the normalized relevance rel_N for a query–result pair using the human relevance judgments that have been persisted in the preprocessing step. For this, all relevance judgments *judg* corresponding to the query–result pair (q,r) are summed up before dividing them by the sum of all judgments for the given query [6]:

$$\mathrm{rel_N}(q,r) = \frac{\sum\limits_{u \in U} \mathrm{judg}(u,q,r)}{\sum\limits_{s \in R} \sum\limits_{u \in U} \mathrm{judg}(u,q,s)} \,, \tag{1}$$

with U the set of users who triggered a judgment and R the set of possible results for the query q. Normalizing judgments is important since otherwise, a result X that was among the results of 20 queries and received 10 positive judgments (rel_N=0.5) would be considered more relevant than a result Y that was among the results of only 5 queries and received 5 positive judgments (rel_N=1).

Having available interaction feature values and normalized relevance of a query–result pair, it is possible to use them as a training instance for SMR's relevance model. For this, the query–result pair is transformed into an instance that can be interpreted by the WEKA API [16]. The interaction features are labeled as attributes while "relevance" is labeled as the target attribute on which we train the model. At the moment, SMR has two built-in classifiers available that are provided by the WEKA API and trained in parallel. That is, a

Hoeffding Tree, which is specifically aimed at incremental learning and is suitable for very large data sets [18], and an updatable version of Naïve Bayes[i], which also works for smaller data sets *(R3)*. The current states of the relevance models are serialized and persisted after each incremental update. These models are ready-to-use *(R3)* and can be instantly engaged for obtaining relevance predictions and feeding them back into a SERP for results optimization (Fig. 2). Moreover, all training instances are persisted to a file to enable manual inspections using, e.g., the WEKA GUI.

Within this topology, emitting a stream of interaction feature values is realized using a *spout (R2)*. Contrary, filtering and normalization tasks as well as incrementally training the relevance models are realized as *bolts*.

The incrementally trained relevance models are serialized and persisted after every update *(R3)*. This makes it possible to manually review the quality of the current model and interrupt or stop training if the model is reasonably stable, which helps to prevent overfitting. Moreover, SMR does not require to directly feed predictions by the incremental relevance model back into the ranking process of the underlying search engine. Rather, as just described, search engine owners are given the option to review the model before usage to ensure ranking quality.

3.4.2 Algorithm

The following pseudocode summarizes the precise functionality of the classification processor:

```
module classificationProcessor {
  /* for simplicity, q := query, r := resultID */
  receive featureValues[q,r]

  if there exists a judgment where q in judgment.queries {
    /* normalize featureValues */
    for each f in features {
      fetch persistedFeatures[q,•] where persistedFeatures[q,•].f is global maximum
      featureValues[q,r].f ← featureValues[q,r].f / persistedFeatures[q,•].f
    }

    compute relevance according to Equation 1

    wekaInstance ← (featureValues[q,r], relevance)
    fetch model
    train model on wekaInstance using WEKA API
    save model
  }
}
```

3.4.3 Industrial Integration

Currently, the search engine's ranking function is applied approximately once a week rather than computing the rank of a result on the fly when a user triggers a query. That is, the ranking score of a result contained in the search engine's index is updated and stored every seven days, which means that at this point in time, the architecture of the search engine is not streaming-oriented and thus not based on Storm [9]. Therefore, it is reasonable to base Turtle on a batch-wise approach, as is also done by TMR.

Turtle engages SMR's approach of incrementally learning relevance models (F. 3, *Model Learning*). That is, we make use of the Naïve Bayes and Hoeffding Tree classifiers, whereas

[i] `http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayesUpdateable.html` (Oct 07, 2013).

only feature values of query–result pairs from the database are added that have not yet been used for learning. Feature values of query–result pairs that have already been considered are flagged accordingly in the database. In case the feature values of a query–result pair are updated by the Interaction Features Processor, the `already_processed` flag is reset to `false`.

As RHiNO crowd workers rate a total of seven attributes for a query–result pair, we can train models to predict each one of these. Moreover, we have introduced a three-class notation of relevance that is determined as follows:

$$\text{relevance}_{q,r} = \begin{cases} \text{good}, & \text{if } N_{q,r}^{good} > N_{q,r}^{bad} \\ \text{neutral}, & \text{if } N_{q,r}^{good} = N_{q,r}^{bad} \\ \text{bad}, & \text{otherwise} \end{cases}, \tag{2}$$

with $N_{q,r}^{good}$ being the number of "good" ratings for result r and corresponding query q etc. Thus, Turtle can provide models for a total of eight target attributes.

Once an up-to-date model is provided by our tool, it is possible to obtain the *Turtle relevance* as an additional weighted feature for the updated ranking function. The turtle relevance is thereby the predicted relevance $\widehat{rel}$ for a query–result pair (q,r) as provided by Turtle's relevance model RM:

$$\text{RM}(q, r, \vec{I}) = \widehat{rel}(q, r)\,, \tag{3}$$

given that interaction feature values $\vec{I}$ are present for (q,r). The new relevance feature is then incorporated into the search engine's ranking function LTR ("learning to rank") as follows:

$$\text{LTR}(q, r) = x_1 \cdot \text{feature}_1(q, r) + \cdots + x_n \cdot \text{feature}_n(q, r) + x_{n+1} \cdot \widehat{rel}(q, r)\,, \tag{4}$$

with x_i being the weightings of the individual features.

The ranking function consists of query-independent as well as query-dependent features. Query-independent features are intrinsic properties of a result—such as the length of the URL—that are independent of the query the result is delivered for. Query-dependent features are specific to a result only for a given query, e.g., the similarity between the query and the result's title. As relevance can only be predicted for a result with a certain query in mind, Turtle relevance belongs to the set of query-dependent ranking features.

When updating the ranking score of a query–result pair, our system checks whether interactions have been tracked in a corresponding search session. If interactions are present, but crowd worker ratings (as delivered by RHiNO) are not, we make use of the Turtle relevance for determining the ranking. Otherwise, we can directly use crowd worker ratings (if present) or omit the new ranking feature.

4 Evaluation

To show SMR's capability of coping with realistic workloads, we have performed a large-scale log analysis of real-world user interactions. The anonymous data used were collected on two large hotel booking portals. We used the *number of conversions* (i.e., when a hotel has been actually booked by users) as relevance judgments for training our models. This stands in contrast to commonly used click models, where clicks are the prime indicators of

relevance. First, we compare SMR to its analogous batch-wise approach TMR (cf. [6]) in terms of the prediction quality of the two systems. Second, we provide BBM (as a state-of-the art generative click model aiming at stream processing; cf. [8]) with the same set of raw interaction logs and compare its quality of relevance prediction against that of SMR. Third, we check SMR against a version of itself that considers clickthroughs only ($\text{SMR}_{\text{click}}$) as well as an analogous version of TMR, i.e., $\text{TMR}_{\text{click}}$. Results indicate that SMR is able to provide reasonably good relevance predictions that are not significantly different from those of TMR and might compare favorably to those of BBM—although the difference is not significant. Moreover, our system is superior to corresponding discriminative approaches that do not consider interactions other than clickthroughs. Fourth, we engage a slightly modified version of our industry system Turtle to investigate the significance of interactions on preceding $(i-1)$ and succeeding $(i+1)$ for predicting the relevance of a result i. Our outcomes indicate that the significance is negligible. Subsequently, we have a look at the efficiency, robustness and scalability of the evaluated approaches. Results show that SMR can easily cope with realistic workloads in a manner that is robust to external influences. This is especially important in real-world settings with big data streams.

For detailed figures and descriptive statistics, see `http://vsr.informatik.tu-chemnitz.de/demo/SMR`. Also, we provide training data and serialized models for reproducing this evaluation using WEKA (cf. [16]).

4.1 Effectiveness

4.1.1 Method

Approximately 23 GB of raw tracking data were collected by SMR's interaction tracking facilities on two large hotel booking portals. Of these, ∼10 GB of interaction logs were chosen for evaluation, which correspond to ∼3.8 million search sessions over a period of 10 days. Based on these, we computed interaction features for a total of 86,915 query–result pairs (cf. Sec. 3.4). Because the collected data contained critical information about the cooperating company's business model, it was a requirement that all data was saved to a key-value store controlled by the company. In particular, we are not allowed to publish the concrete conversion–to–search session (CTS) ratio. Yet, it can be stated that this ratio is very low, i.e., $\#\textit{conversions} \ll \#\textit{search sessions}$.

We divided the chosen raw interaction data into *10 distinct data sets DS0–DS9* (∼0.7–1.5 GB each) that were intended for training relevance models and corresponded to one day each. Since SMR cannot—due to its streaming-based nature—use fixed maximum values for interaction feature normalization (cf. Section 3.4), it produces different feature values for the same tracking data compared to TMR. Thus, processing the above raw data sets with both systems yields a total of 20 data sets containing interaction features and relevances (i.e., normalized conversions) of the extracted query–result pairs: DS^0_{TMR}–DS^9_{TMR} from TMR and DS^0_{SMR}–DS^9_{SMR} from SMR. For this, we considered only search sessions that were produced by users who triggered at least one conversion (in terms of booking a hotel). Conversions are treated as relevance judgments in analogy to [6], i.e., a greater number of conversions implies higher relevance and vice versa. For evaluating SMR, we *simulated a stream* of search sessions based on the logs containing raw interaction data.

Table 1. Evaluation of two exemplary models highlighting that the MCC is a more suitable measure for our evaluation.

	TN	FP	FN	TP	F-measure	MCC
DS^1_{SMR}	13413	115	362	48	0.959	0.171
DS^4_{SMR}	2944	39	120	28	0.940	0.258

The Storm cluster used for evaluation was based on *Amazon EC2*[j]. It comprised four computing instances. An additional machine was used for logging purposes and hosting the database used. All computers in the Storm cluster were instances of type `m1.large`, featuring two CPUs and 7.5 GB RAM[k].

4.1.2 Matthews Correlation Coefficient

In analogy to [6], we observed a very low ratio of booked hotels to search sessions. In addition with a high query diversity this leads to more than 99% of the query–result pairs having a relevance of either 0.0 or 1.0. Therefore, in this evaluation, we treat relevance prediction as a binary classification problem with two classes: "bad" (relevance < 0.5) and "good" (relevance ≥ 0.5). With more than 90% of the query–result pairs having a *bad* relevance and less than 10% having a *good* relevance, these classes are rather unbalanced. Thus, we use the *Matthews Correlation Coefficient* (MCC) for evaluations of model quality, which is particular suitable for cases with unbalanced classes [17].

For a binary classification problem, the Matthews Correlation Coefficient is defined as [17]:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \,. \tag{5}$$

The MCC is based on numbers of TP = true positives, TN = true negatives, FP = false positives and FN = false negatives and is therefore related to the *F-measure*, which is given by $F = 2 \times \frac{precision \times recall}{precision + recall}$ with $precision = \frac{TP}{TP+FP}$ and $recall = \frac{TP}{TP+FN}$.

Yet, the MCC is a more balanced measure for highly unbalances classes [17]—as is the case in this evaluation—, which is demonstrated in Table 1 based on *Random Forest* models trained for two representative data sets of different sizes. Thus, in the following, we are going to report MCC rather than F-measure values, which are skewed and unrealistically high since $TN \gg FP + FN + TP$ for all data sets.

4.1.3 SMR vs. TMR

Based on the data sets described above, we trained a total of 20 Naïve Bayes classifiers (10 per system), as provided by TMR and SMR through the WEKA API[l]. Thereby, our system used the updatable version of the classifier for incremental learning. The Naïve Bayes classifier was chosen because the amount of data available for evaluation was too small to train reasonably good Hoeffding Tree classifiers [18]. All classifiers learned have been evaluated using *10-fold cross validation*, from which we obtained corresponding MCC values. As can be seen in Fig. 6, the difference between SMR and TMR is not significant across the 10 data sets. This result

[j] `http://aws.amazon.com/ec2` (Sep 30, 2013).

[k] `http://aws.amazon.com/en/ec2/instance-types/#instance-details` (Oct 05, 2013).

[l] Relevance models provided by SMR and TMR are highly sensitive to layout specifics of the corresponding SERPs [6]. Yet, since the two hotel booking portals feature the exact same layout template, it is valid to use combined data from both portals for training the same model(s).

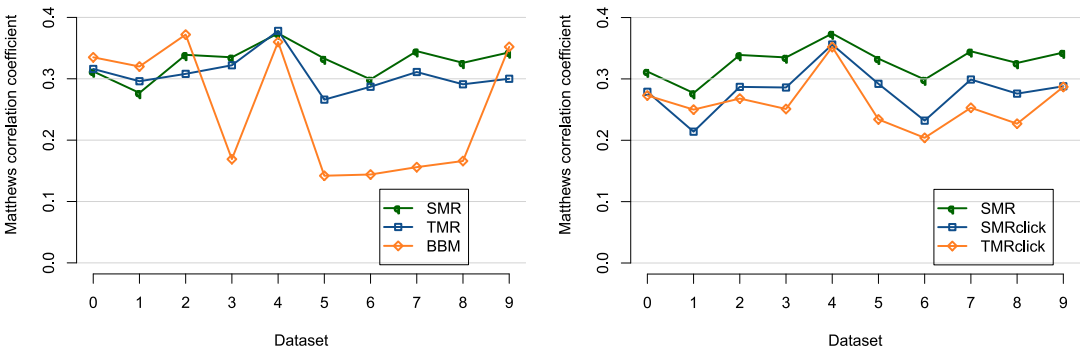

Fig. 6. MCC values for *DS0–DS9* (threshold = 0.5).

has been validated using a Wilcoxon rank sum test, with $p>0.05$ (α=0.05, W=75, 95.67% conf. int. = [-0.047, 0.004]). It implies that statistically, SMR yields the same prediction quality as TMR, even though it has less information available; particularly in terms of feature normalization and missing look-ahead capabilities. While Fig. 6 shows only MCC values at a threshold of 0.5, our result is underpinned by the exemplary receiver operating characteristic (ROC) curves depicted in Fig. 7, where SMR does not dominate TMR or vice versa. TMR performs slightly better in terms of its true positive rate for small false positive rates ($\lesssim 0.15$), while SMR does so for false positive rates $\gtrsim 0.15$. Yet, the *areas under ROC* lie close together ($\text{AUROC}_{\text{TMR}} = 0.850$, $\text{AUROC}_{\text{SMR}} = 0.861$), which shows that both systems' predictions are considerably better than chance (AUROC = 0.5), but prediction quality is not significantly different. These results are similar for the remaining nine data sets.

4.1.4 SMR vs. BBM

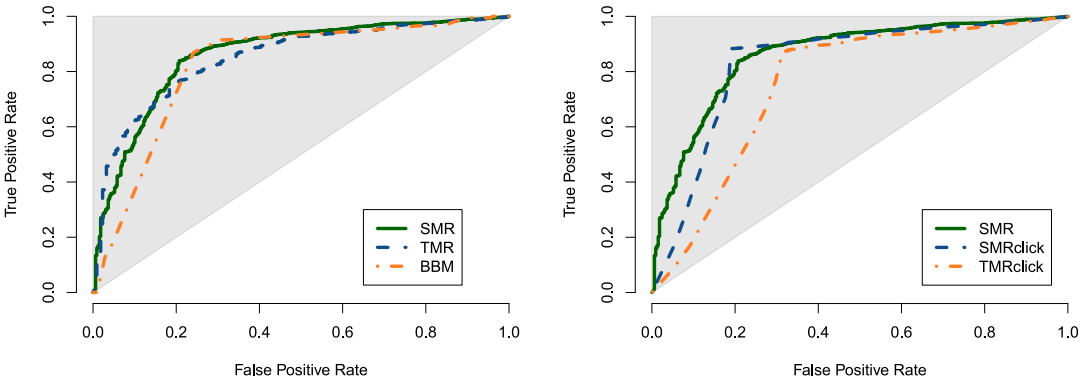

Fig. 7. ROC values for *DS7*.

Additionally, we have compared SMR's prediction quality to that of a state-of-the-art generative click model designed for very large amounts of data and incremental learning. For this, we have used an existing re-implementation of BBM—as described in [8]—and provided

it with the exact same raw interaction logs. Fig. 6 shows that BBM yields slightly better predictions for four out of ten data sets (*DS0–DS2*, *DS9*) at a threshold of 0.5 while SMR has a better prediction quality for the remaining six data sets. For this, predictions of BBM have been compared to the normalized relevances computed by SMR based on the available conversions. The difference between the two approaches is not significant according to a Wilcoxon rank sum test (α=0.05, W=64.5, p>0.05, 95.67% conf. int. = [-0.177, 0.021]). Still, our result indicates that SMR has the potential to provide relevance predictions that compare favorably to BBM. Particularly, Fig. 7 suggests that predictions of BBM can be partly dominated by SMR's predictions for certain data sets. That is, for the exemplary data set *DS7*, BBM has a slightly better true positive ration than SMR only for a small interval around a false positive rate of ≈ 0.25. Also, with a value of 0.826 the AUROC of BBM is considerably better than chance, but lower compared to both, SMR (0.861) and TMR (0.850). Since we expect SMR's prediction quality to increase with amounts of data larger than used in this evaluation, we hypothesize that our system can predict relevance at least as good as BBM, whose predictions are being successfully used in industry.

4.1.5 SMR vs. SMR_{click} vs. TMR_{click}

Table 2. Wilcoxon rank sum tests for SMR vs. SMR_{click} and SMR vs. TMR_{click}.

	α	W	p	95.67% confidence interval
SMR vs. SMR_{click}	0.05	84.5	<0.05	[-0.075, -0.020]
SMR vs. TMR_{click}	0.05	90.0	<0.01	[-0.101, -0.044]

To investigate the influence of the additional user interactions, we have performed a comparison of SMR to versions of itself and TMR that consider clickthroughs only, named SMR_{click} and TMR_{click}. Results show that SMR outperforms the click-only approaches across all 10 data sets (Fig. 6) based on *10-fold cross-validation*. Moreover, the MCC differences between SMR and SMR_{click}/TMR_{click} are significant, as has been shown by two Wilcoxon rank sum tests (Table 2). Our results are further supported by the ROC curves shown in Fig. 7, where SMR (AUROC = 0.861) performs better than both SMR_{click} (AUROC = 0.834) and TMR_{click} (AUROC = 0.759). In fact, the ROC curve of TMR_{click} is dominated by that of SMR across all thresholds, particularly clearly for false positive rates $\lesssim 0.3$. Moreover, SMR_{click} shows a better true positive ratio compared to SMR only for a small interval around a false positive rate of ≈ 0.2. These findings underpin that adding interaction data other than clicks yields considerable improvements for discriminative approaches, as has also been outlined in [3, 5]. This is true even if clickthroughs show a correlation with relevance that is notably higher than those of the additional attributes (e.g., r=0.34 for DS^2_{TMR}).

4.1.6 Interactions on Preceding/Succeeding Results

In [6, Sec. 3.5.3], Speicher et al. investigate correlations between mouse features of a search result and its relevance. Following this, we have engaged a slightly modified version of Turtle to reprocess a fraction of the data used in [6]. Unfortunately, the novel search engine—into which Turtle has been integrated—had not yet delivered a sufficient amount of high-quality tracking data; therefore we had to rely on existing data sets. With these, we have investigated correlations of the relevance of a result i with mouse features of the preceding $(i-1)$ and succeeding $(i+1)$ results. Particularly, we used data sets DS1 and DS2, as described in [6].

While DS2 could be completely reprocessed, due to technical limitations we were only able to reprocess approximately ⅔ of DS1 (which we therefore refer to as DS1′). Yet, this does not affect the generality of our results (in the context of the data used). Correlations are given in Table 3.

Table 3. Correlations between relevance of result i and mouse features of results $i-1$, i and $i+1$.

	DS1′			DS2			combined		
Pearson's r	$i-1$	i	$i+1$	$i-1$	i	$i+1$	$i-1$	i	$i+1$
avg. hover time	0.02	0.20	0.02	0.02	0.19	0.02	0.02	0.20	0.02
arrival time	0.06	0.14	0.03	0.07	0.14	0.06	0.07	0.14	0.05
clicks	0.03	0.07	0.03	0.00	0.09	-0.01	0.01	0.09	0.00
clickthroughs	0.01	**0.42**	0.03	0.01	0.35	-0.01	0.01	0.37	0.00
hovers	0.00	0.16	-0.02	0.01	0.16	-0.01	0.01	0.16	-0.01
max. hover time	0.03	0.20	0.03	0.03	0.19	0.03	0.03	0.20	0.03
cursor trail	0.01	0.06	-0.01	0.00	0.06	0.02	0.00	0.06	0.01
cursor move time	0.03	0.18	0.04	0.03	0.19	0.03	0.03	0.19	0.03
cursor speed	**0.10**	0.13	**0.09**	0.07	0.14	0.06	0.08	0.14	0.07
#instances	5439	8969	5400	10514	19875	10381	15953	28844	15781

Our analysis was based on the assumption that if less or more interactions happen on preceding/succeeding results, this should be an indicator of how interesting result i appears to the user—and thus also of its relevance. However, as can be seen, there exist no considerable correlations between mouse features of results $i-1$ and $i+1$ and the relevance of result i. The greatest correlations are 0.10 for cursor speed on result $i-1$ and 0.09 for cursor speed on result $i+1$, as opposed to a correlation of 0.42 between clickthroughs on result i and its relevance. The correlations of mouse features of result i are of the order of the numbers given in [6].

Our results indicate that users seem to be treating search results as independent entities on SERPs. However, this finding holds only for the interaction data collected in the specific travel search setting described in [6]. Thus, additional analyses with data from general search settings should be conducted. In particular, the above correlations show differences compared to [3], who investigated such a more general setting.

4.2 Efficiency, Scalability and Robustness

4.2.1 Efficiency and Scalability

SMR is a feasible approach for processing web-scale interaction data. In contrast, TMR uses a batch-wise approach and non-incremental classifiers [6]. This means that all training data (in terms of query–result pairs, i.e., interaction features and relevances) already put into a model have to be reprocessed for an update. For training an up-to-date model, this yields a time-complexity of $O(c(q+q'))$ with c = complexity of adding one instance to the classifier used, q = #query–result pairs in new search sessions since last processing and q' = #previously processed query–result pairs. Assume we receive one log with raw interaction data per day and want a daily model update. Then the amount of data that needs to be reprocessed grows linearly. At some point, processing these data would take longer than 24 hours unless we add more/faster hardware to the system, which is, however, not a feasible approach in the long-term. Particularly, reprocessing previously processed query–result pairs involves numerous slow database requests. To give just one concrete example from our evaluation, TMR needs ∼5 hours for processing a single 1.5 GB log on a dual-core machine with a 2.3 GHz *Intel Core*

i5 CPU and 4 GB RAM. Since this corresponds to one day, processing the logs for two days would already take ∼10 hours etc. This means that after *five days*, we exceed a processing time of 24 hours, which makes it impossible to provide a daily model update unless we use a better machine than the given one.

In contrast, SMR does not need to reprocess logs from previous days since data is processed on a per–search session basis and models are learned incrementally. Thus, a model update considers only one search session at a time and the time-complexity of the update is q times the complexity of the classifier used, with q = #query–result pairs in search session. For instance, "constant time per example [i.e., a query–result pair in our case]" [18] if using a Hoeffding Tree, which would be $q \times O(1) = O(q)$. The time-complexity for preprocessing the raw data of a search session is the same for TMR and SMR. That is, $O(e + qu)$, with e = #events in search session, q = #query–result pairs in search session and u = complexity of updating a DB entry. SMR needs ∼2 hours for processing all search sessions in a 1.5 GB log using the cluster described in Section 4.1.1. For this, the search sessions have been put into the system at the highest possible frequency. The log used corresponds to one day of real-world traffic from two hotel booking portals. This means that—using simple interpolation—SMR would be able to cope with approximately *12 times the load* based on the relatively simple cluster set-up used.

Finally, BBM has been specifically designed for incremental updates and web-scalability. As described in [8], 0.25 PB of data were processed using the generative click model. The authors state that it was possible to compute relevances for 1.15 billion query–result pairs in three hours on a MapReduce [19] cluster. BBM's time-complexity for updating a relevance model is $O(s)$, with s = #new search sessions since last processing.

Due to the differences in system architecture—TMR runs on a single node while the other two approaches require a cluster—the above is not an absolute, hardware-independent comparison of performance. Rather, it describes *relative performances* between the three systems. An overall, relative comparison of efficiency and scalability of the compared approaches is shown in Table 4.

4.2.2 Robustness

Being based on Storm, SMR is a highly robust system by design. In particular, it features guaranteed message passing[m] and high fault-tolerance[n] if one or more nodes die due to external reasons—which happened numerous times during our evaluation. In such a case, SMR continued processing the current interaction data from the step prior to the incident.

In [8], Liu et al. do not explicitly address the robustness of their approach. Rather, BBM has been designed for use as a MapReduce job on a *Hadoop* cluster. That is, differences in robustness between SMR and BBM originate from corresponding differences between Storm and Hadoop. Particularly, Hadoop has disadvantages when it comes to guaranteed message processing or when supervising/master nodes are killed.

Finally, TMR is the least robust of the compared approaches. In case the processing of a batch of data is stopped due to external reasons (e.g., a memory overflow), all data need to be reprocessed. In particular, this means that already computed values of interaction features

[m] `https://github.com/nathanmarz/storm/wiki/Guaranteeing-message-processing` (Dec 30, 2013).
[n] `https://github.com/nathanmarz/storm/wiki/Fault-tolerance` (Dec 30, 2013).

are useless since contributions of already processed data cannot be subtracted out before starting over an iteration. Therefore, careful evaluation and set-up of the required hardware are necessary before using TMR to minimize the risk of costly and time-consuming errors.

4.3 Discussion and Summary

In this evaluation, we have shown that SMR does not perform significantly less effective than TMR, even though it relies on lower-quality information for training its relevance models. Moreover, SMR is more efficient, robust and scalable compared to its batch-wise predecessor. The difference of SMR's predictions to those of the generative state-of-the-art click model BBM were not significant as well. Yet, our results indicate that our discriminative approach can be advantageous over BBM for certain data sets and that it is more robust at similar efficiency and scalability. Finally, we have underpinned the value of interaction data other than clicks for relevance prediction, with clickthrough-only versions $\text{SMR}_{\text{click}}$ and $\text{TMR}_{\text{click}}$ performing significantly worse than SMR. However, there are some points remaining for discussion.

4.3.1 Discussion

Why does SMR show the tendency to perform better than TMR, although its training data are of lower quality? As described in Section 3.4, the maximum values for feature normalization change during the processing of a data set due to SMR's streaming-based nature (i.e., no look-ahead is possible). This means that SMR has less information available and as a result, the training data has lower quality. However, the different feature values for query–result pairs that appear early in a data set can—purely by chance—lead to better predictions of SMR. This is especially the case because in this evaluation we were working with relatively small and closed data sets, as compared to a real-world setting. Hence, we strongly assume that in such a setting, the already non-significant difference between SMR and TMR would become even smaller.

Why does BBM make better predictions than SMR for DS2 but predicts worse for DS7? SMR computes almost the same amount of query–result pairs for the two data sets, with nearly identical means and distributions of the individual interaction features. In contrast, BBM has approximately 12% less search sessions available in *DS7* compared to *DS2*, which is due to the fact that search sessions are treated differently by BBM. Our system treats every page load event on a SERP as the beginning of a new search session. That is, if a user clicks a result and then returns to the SERP for clicking another result, SMR interprets this as two separate sessions. However, BBM handles this as a single search session with two clickthrough events. Besides containing more of these "combined" search sessions, *DS7* also features ∼12% less clickthrough events. All in all, this results in BBM having less data available for training its relevance model, which is an explanation for the lower-quality prediction compared to *DS2*. The same holds for other data sets showing similar differences, *DS2* and *DS7* are only used for representative purposes here.

Why are the MCC values relatively low (< 0.5) in general? The data collected for evaluation featured a very low CTS ratio, i.e., the amount of interaction data exceeded the available relevance judgments by far. To give just one example, the CTS ratios of both *DS0* and *DS1* lie under 1%, which is similar for the remaining data sets. This and the fact that the data sets used for evaluation were relatively small (compared to a realistic long-term scenario) leads to a rather low data quality. Yet, in an evaluation with larger amounts of data, we would expect

increasing MCC values. Particularly, Huang et al. state that "adding more data can result in an order of magnitude of greater improvement in the system than making incremental improvements to the processing algorithms" [7].

How does SMR deal with click spam? Click spam is a major problem in systems where clicks are the main indicator for relevance [20]. However, in the specific setting we are focusing on in this paper, a high number of *conversions* indicates high relevance. Since conversions imply a confirmed payment, we do not have to deal with "traditional" click spam as described in [20]. Yet, in settings where no conversions are available, our discriminative approach has to rely on other indicators of relevance, such as clicks on social media buttons, for training its models. In such cases, additional measures have to be taken that prevent fraudulent behavior aiming at manipulating relevance models. Potential measures could be based on, e.g., filtering pre-defined behavior profiles, blacklists, personalized search [20] or the ranking framework described by [21].

4.3.2 Summary

Table 4. Overall relative comparison of the considered approaches.

	effectiveness	efficiency	robustness	scalability
SMR	0	++	++	++
BBM	−	++	+	++
TMR (baseline)	*0*	*0*	*0*	*0*
$\text{SMR}_{\text{click}}$	−−	++	++	++
$\text{TMR}_{\text{click}}$	−−	0	0	0

Table 4 shows a comparison of all approaches considered in the evaluation. Since the systems—due to differences in the underlying architectures—are difficult to compare in an absolute, hardware-independent manner, we give a comparison of relative performances. Using TMR as the baseline, "0" indicates similar performance, "+"/"−" indicate a tendency and "++"/"−−" indicate a major or significant difference.

Table 5. High-level comparison of TMR, SMR and the hybrid solution Turtle.

	raw data processing	feature computation	incremental models
TMR [6]	batch-wise	batch-wise	∅
SMR	real-time (Storm)	real-time (Storm)	✔
Turtle	batch-wise (cron job)	batch-wise (cron job)	✔

Moreover, a high-level comparison of different aspects of TMR, SMR and Turtle is given in Table 5.

5 Limitations and Future Work

The following section discusses the limitations of (a) the novel streaming-based system SMR as well as (b) the more specific industry solution Turtle, and provides an overview of potential future work.

5.1 SMR

As described in this article, SMR specifically aims at relevance prediction in the context of *travel search*. One specific feature of this setting is the fact that we can use hotel booking conversions as indicators of relevance. However, in a more general setting, other implicit or

explicit relevance judgments are necessary. For example, one could obtain such judgments by providing optional vote up/down buttons to visitors or tracking clicks on Facebook "Like" buttons of a search result. Hence, we have transferred SMR into a real-world industry context with a more general search setting. For this, we make use of an additional crowdsourcing tool that delivers relevance judgments produced by internal crowd workers. The limitations of the resulting new system Turtle are described in the following section.

Concerning the evaluation of our system, we had to rely on relatively small data sets compared to the real-world settings the system is intended for in the long-term. As part of our future work, we intend to evaluate SMR with larger data sets that simulate a real-world setting of a time-span considerably longer than 10 days. This will also give us the chance to investigate the performance of the Hoeffding Tree classifier, which becomes feasible only for very massive amounts of data [18].

Currently, SMR is only able to track client-side interactions on desktop PCs, i.e., mouse input. However, since the mobile market is steadily growing, an increasing number of users access search engines using their (small-screen) touch devices. This demands for also making use of touch interactions for predicting the relevance of results. Leveraging these valuable information is especially important for search engine owners and intended in future versions of SMR.

Finally, interaction features are often coupled with temporal features or their values change over time. This has to be addressed in the context of *concept drift* [22]. SMR is generally capable of handling changing data streams, as Tsymbal states that "[i]ncremental learning is more suited for the task of handling concept drift" [22]. However, the Naïve Bayes classifier used in the context of this paper would have to be replaced by an adequate concept drift–ready learner. A potential candidate is the CVFDT learner, which is based on Hoeffding trees and dismisses a subtree based on old data whenever a subtree based on recent data becomes more accurate [23].

5.2 Turtle

In Section 3, we have described Turtle—a hybrid TMR/SMR solution that serves as an industrial use case. It uses crowdsourced relevance judgments for learning corresponding models. Turtle is a pragmatic approach tailored to the needs of the novel search engine it has been integrated into.

Yet, the system still has several shortcomings. First, a company-internal crowdsourcing tool is not optimal for obtaining a maximum possible number of human relevance judgments, as the number of crowd workers is rather limited. Therefore, as part of future work, it would be desirable to integrate our solution with *Amazon Mechanical Turk*[o] to attract a larger number of crowd workers. This, in return, would be less optimal concerning the cost factor from the company's point of view, which makes it necessary to find a trade-off between number of crowd workers and costliness. Yet, over a course of approximately two weeks, RHiNO was able to deliver ratings for 14,378 query–result pairs.

Second, a query–result pair whose feature values have been considered for learning a relevance model might be updated when new search session data are available. Thus, it has to be considered again in the next iteration of incrementally updating the model. This

[o] `https://www.mturk.com/mturk/welcome` (Oct 23, 2014).

means that interaction feature values become more representative over time and relevance models require a decent amount of training data for delivering good predictions. This holds particularly if the Hoeffding Tree classifier is used.

Finally, we were not yet able to meaningfully evaluate Turtle, as the novel search engine is still in a closed beta state. Thus, the interaction data tracked so far is not of a large enough size and good enough quality (interaction features for 1,516 query–result pairs, as opposed to 86,915 query–result pairs in the evaluation of SMR). However, we are determined to report on this as part of future work.

6 Conclusions

This article introduced SMR, which is a novel approach to providing incremental models for predicting the relevance of web search results from real-time user interaction data. Our approach helps to ensure one of the *prime aspects of search engine quality*, i.e., providing users with the most relevant results for their queries. In contrast to numerous existing approaches, SMR does not require reprocessing of already processed data for obtaining an up-to-date relevance model. Moreover, our system involves interaction features other than clicks and was specifically designed for coping with large amounts of data in real-time. This allows for feeding relevance predictions back into SERPs with relatively low latency.

For evaluating SMR, we have simulated a *real-world setting* with large amounts of interaction data from two *large hotel booking portals*. Comparison of our system to the analogous batch-wise approach TMR showed that SMR is able to predict relevances that do not differ significantly, although it has less information available for training. Furthermore, we have compared the discriminative SMR approach to BBM—a generative state-of-the-art click model for incrementally processing big data streams that is successfully applied in industry. Results show that prediction quality does not differ significantly between the two systems. Still, they indicate that predictions by SMR might compare favorably to those of BBM, as it outperforms the click model for the majority of data sets. Additionally, we have considered a click-only version of SMR that was compared to the complete system. From the significantly better predictions of the latter, we conclude that interactions other than clicks yield valuable information for relevance prediction and should not be neglected.

Additionally, we have transferred our system into a *real-world industry context* that focuses on general web search rather than a hotel booking setting. For this, we have engaged a hybrid solution—named Turtle—comprising parts of both, TMR and SMR, as the novel search engine does not yet build on a streaming-based approach. To obtain relevance judgments for learning corresponding models, we have furthermore developed a new crowdsourcing tool for rating search results w.r.t. a given query. Using Turtle, it is possible to incorporate relevance predictions based on user interactions into the process flow of the novel search engine. That is, predictions are used as a weighted feature of the global ranking function.

As future work, it is planned to further optimize the system regarding performance and perform an evaluation with even larger amounts of real-world interaction data. Moreover, we intend to also focus on touch interactions rather than limiting our system to the desktop PC setting. Finally, future work will include an evaluation of Turtle, which to date was not possible due to a lack of sufficient high-quality tracking data.

Acknowledgements

Special thanks go to Christiane Lemke and Liliya Avdiyenko for supporting us with their implementation of BBM. We have used free icons by `http://www.vectortown.com/`. This work has been partially supported by the ESF and the Free State of Saxony.

References

1. N. Craswell, O. Zoeter, M. Tylor and B. Ramsey (2008), *An Experimental Comparison of Click Position-Bias Models*, Proc. WSDM, pp. 87–94.
2. Q. Guo and E. Agichtein (2012), *Beyond Dwell Time: Estimating Document Relevance from Cursor Movements and other Post-click Searcher Behavior*, Proc. WWW, pp. 569–578.
3. J. Huang, R.W. White and S. Dumais (2011), *No Clicks, No Problem: Using Cursor Movements to Understand and Improve Search*, Proc. CHI, pp. 1225–1234.
4. V. Navalpakkam and E.F. Churchill (2012), *Mouse Tracking: Measuring and Predicting Users' Experience of Web-based Content*, Proc. CHI, pp. 2963–2972.
5. J. Huang (2011), *On the Value of Page-Level Interactions in Web Search*, Proc. HCIR Workshop.
6. M. Speicher, A. Both and M. Gaedke (2013), *TellMyRelevance! Predicting the Relevance of Web Search Results from Cursor Interactions*, Proc. CIKM, pp. 1281–1290.
7. J. Huang, R.W. White, G. Buscher and K. Wang (2012), *Improving Searcher Models Using Mouse Cursor Activity*, Proc. SIGIR, pp. 195–204.
8. C. Liu, F. Guo and C. Faloutsos (2009), *BBM: Bayesian Browsing Model from Petabyte-scale Data*, Proc. KDD, pp. 537–546.
9. A. Both, A.-C. Ngonga Ngomo, R. Usbeck, D. Lukovnikov, C. Lemke and M. Speicher (2014), *A Service-oriented Search Framework for Full Text, Geospatial and Semantic Search*, Proc. SEMANTiCS, pp. 65–72.
10. N. Marz, *Storm Wiki*, `https://github.com/nathanmarz/storm/wiki`, retrieved October 15, 2014.
11. M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker and I. Stoica (2012), *Discretized streams: A fault-tolerant model for scalable stream processing*, Technical Report, UC Berkeley.
12. T. Joachims (2002), *Optimizing Search Engines using Clickthrough Data*, Proc. KDD, pp. 133–142.
13. F. Guo, C. Liu and Y.M. Wang (2009), *Efficient Multiple-Click Models in Web Search*, Proc. WSDM, pp. 124–131.
14. O. Chapelle and Y. Zhang (2009), *A Dynamic Bayesian Network Click Model for Web Search*, Proc. WWW, pp. 1–10.
15. G.E. Dupret and B. Piwowarski (2008), *A User Browsing Model to Predict Search Engine Click Data from Past Observations*, Proc. SIGIR, pp. 331–338.
16. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I.H. Witten (2009), *The WEKA Data Mining Software: An Update*, SIGKDD Explor. Newsl., 11(1), pp. 10–18.
17. P. Baldi, S. Brunak, Y. Chauvin, C.A. Andersen and H. Nielsen (2000), *Assessing the accuracy of prediction algorithms for classification: an overview*, Bioinformatics, 16(5), pp. 412–424.
18. P. Domingos and G. Hulten (2000), *Mining High-Speed Data Streams*, Proc. KDD, pp. 71–80.
19. J. Dean, S. Ghemawat (2008), *MapReduce: Simplified Data Processing on Large Clusters*, CACM, 51(1), pp. 107–113.
20. F. Radlinski (2007), *Addressing Malicious Noise in Clickthrough Data*, Proc. LR4IR@SIGIR.
21. J. Bian, Y. Liu, E. Agichtein and H. Zha (2008), *A Few Bad Votes Too Many? Towards Robust Ranking in Social Media*, Proc. AIRWeb, pp. 53–60.
22. A. Tsymbal (2004), *The problem of concept drift: definitions and related work*, Technical Report, Trinity College Dublin.

23. G. Hulten, L. Spencer and P. Domingos (2001), *Mining Time-Changing Data Streams*, Proc. KDD, pp. 97–106.
24. M. Speicher, S. Nuck, A. Both and M. Gaedke (2014), *StreamMyRelevance! Prediction of Result Relevance from Real-Time Interactions and its Application to Hotel Search*, Proc. ICWE, pp. 272–289.

Author Statement

This article is an extended version of [24], which has been chosen as one of the best papers of the 2014 *International Conference on Web Engineering* (ICWE '14). We have augmented the original research paper with an in-depth explanation of SMR's algorithmic contribution, e.g., the finite state machine defining valid sequences of mouse cursor events. Moreover, we describe a new SMR-based hybrid solution named Turtle that has been integrated into a real-world search engine. A modified version of Turtle has been used to extend the original evaluation by investigating the significance of interactions on preceding/succeeding results for predicting relevance. Also, we give more details about the specific time-complexity of our system and competing approaches. As a minor extension, we present a scenario from which three requirements for the system are derived.