StreamMyRelevance!
Prediction of Result Relevance from Real-Time Interactions and Its Application to Hotel Search

Maximilian Speicher[1,2], Sebastian Nuck[2,3], Andreas Both[2], and Martin Gaedke[1]

[1] Chemnitz University of Technology, 09111 Chemnitz, Germany
[2] R&D, Unister GmbH, 04109 Leipzig, Germany
[3] Leipzig University of Applied Sciences, 04277 Leipzig, Germany
`maximilian.speicher@s2013.tu-chemnitz.de`, `martin.gaedke@informatik.tu-chemnitz.de`, `andreas.both@unister.de`, `sebnuck@gmail.com`

Abstract. 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 updateable 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.

Keywords: Streaming, Real-Time, Interaction Tracking, Learning to Rank, Relevance Prediction.

1 Introduction

Nowadays, search engines are among the most important and most popular web applications. They are essential for supporting users with finding specific pieces of information on the web. Thus, their *prime aspect of quality* is to ensure that relevant results are displayed where they receive the highest attention. In other words, the ranking of results is a major quality aspect in the context of the search application as a whole. This makes it necessary to estimate the relevance of results *a priori*. Common methods for obtaining such estimates are generative *click models* (e.g., [3,4,15]). Based on certain assumptions about user behavior, these models predict the relevance of a certain result taking into account the number of clicks it has received for a given query. However, click data are not a perfect indicator concerning relevance since users might return to the search engine results page (SERP) after having clicked a useless result. Additionally, search engines more and more try to answer queries directly on the SERP, e.g., as Google do with their *Knowledge Graph*[1]. 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 [9] or other client-side user behavior (e.g., [9,13,18]). Previous research has shown the value of such page-level interactions [11,13,20]. Also, generative [12] as well as discriminative [20] approaches to relevance prediction exist that engage user behavior other than clicks only.

With a growing amount of users, it is possible for search engine providers to collect enormous amounts of client-side data. This is particularly the case if we consider interactions other than clicks. Along with the increasing quantities of tracking data, a short time-to-market becomes more and more important. That is, providers need to quickly analyze collected information and feed potential findings back into their products to ensure user satisfaction. This calls for the use of novel systems for data stream mining, such as *Storm*[2], which are currently gaining popularity in industry and research. These systems can help to cope with the seemingly endless streams of data produced by today's internet users. Yet, none of the approaches for relevance prediction mentioned above leverages data stream mining to process collected information.

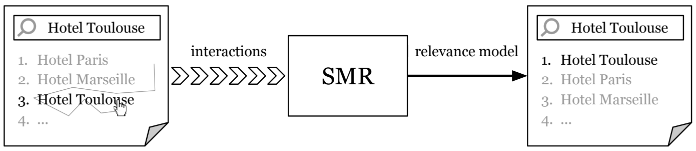

Fig. 1. The intention behind StreamMyRelevance!—from collecting a stream of user interactions to reordering search results based on relevance models

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 to learning incremental *relevance models*, i.e., models that predict the relevance of a search result (for a given query) based on certain features of user interaction. 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 [20]. Yet, TMR is a batch-oriented approach that does not provide means for incrementally learning relevance models on

[1] `http://www.google.com/insidesearch/features/search/knowledge.html` (2013-09-06).
[2] `http://www.storm-project.net/` (2013-12-30).

a streaming basis. Thus, SMR wraps the borrowed functionalities into a new system that is able to handle real-time streams. Our system has three main advantages over existing approaches, i.e., (1) considering interactions other than clicks for predicting relevance, (2) collecting and processing these interactions as a stream and (3) providing incremental relevance models that do not require re-processing of previously processed data. Based on this, the main hypothesis investigated in this paper is 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, large amounts of real-world data from two *hotel booking portals* were available. A comparison to TMR has been performed, which due to its batch-oriented design has look-ahead capabilities and thus more information available [20]. 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) [16], 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.

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, 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 [17], 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[3]. It as well provides distributed stream computing functionality, but its underlying concepts and configuration are more complex[4]. As described in [22], benchmarks have shown that S4 is almost 10 times slower than Storm.

This 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.

[3] `http://incubator.apache.org/s4/` (2013-09-28).
[4] `http://demeter.inf.ed.ac.uk/cross/docs/s4vStorm.pdf` (2014-01-06).

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 [15] proposes to use *clickthrough data* instead of human relevance judgments. Based on the *cascade hypothesis* [4,16], 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 [15]. In contrast to the above, models like the *Dependent Click Model* [8] 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 [3] generalizes the *Cascade Model* [4] 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 [16]. 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 SMR are that we aim at predicting relevance using a *discriminative* approach also taking into account *interactions other than clicks*. Moreover, the above click models are not designed for efficient processing of *massive data streams* or *incremental updates*.

The *Bayesian Browsing Model* (BBM) [16] is based on the *User Browsing Model* (UBM) [7], which assumes that the probability of examination depends on the position of the last click and the distance to the current result [16]. 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" [16]. 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.

Concerning user interactions other than clicks, in [11], Huang has found that these are a valuable source of information for relevance prediction. Following, Huang et al. [13] investigate the correlations between human relevance judgments and mouse features such as hover time and unclicked hovers, among others. They find positive correlationsand 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 [13].

In [9], 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 [3], 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. [12] 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.

TMR is a system described by Speicher et al. [20]. 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. In particular, its relevance models are not trained incrementally, i.e., *all* data have to be re-processed before obtaining an updated model.

3 SMR: Streaming Interaction Data for Learning Relevance Models

The following Section describes SMR, which is organized as a streaming-based process. 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

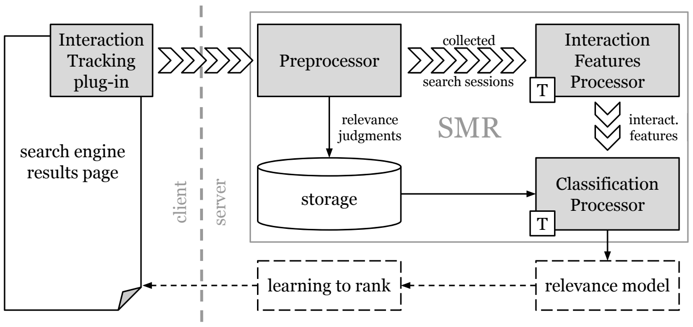

Fig. 2. The main components and process flow of SMR (Streams are visualized by sequences of chevrons; Storm topologies are annotated using a "T")

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.

3.1 Client-Side Interaction Tracking

For client-side interaction tracking, SMR builds upon a "minimally invasive jQuery plug-in" [20] that is provided by TMR. This plug-in tracks `mouseenter`, `mousepause`, `mousestart`, `mouseleave` and `click` events that happen within the bounds of a search result on a SERP [20]. Each mouse event is extended with the search query, a user ID and the ID of the corresponding result [20]. The resulting data packets are then sent to a specified key-value store at suitable intervals (Fig. 2) [20]. For integration, the developer has to specify jQuery selectors 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.

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 [20]. 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.

3.2 Preprocessor

After having been recorded using the above jQuery plug-in, all interaction data is received by SMR as a *stream of individual events* for preprocessing (Fig. 2). Additionally, information about a corresponding search session[5] 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 [20]. 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. Since it is unreliable to fire client-side `unload` events on

[5] 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.

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.

3.3 Interaction Features Processor

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. To ensure that all interaction events associated with a search session are ordered logically, invalid sequences of events are filtered out. 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.

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 are:

(i) Arrival time, (ii) clicks (not leading to a landing page), (iii) clickthroughs (leading to a landing page), (iv) cursor movement time, (v) cursor speed (cursor trail divided by cursor movement time), (vi) cursor trail, (vii) hovers, (viii) hover time, (ix) maximum hover time, (x) position and (xi) unclicked hovers (hovers during which no clickthrough happened).

These are in accordance with [20]. 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* [20]. 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.* 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.

3.4 Classification Processor

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. 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 [20]. Thus, normalization of feature values is necessary to guarantee comparability between models related to different SERP layouts [20]. 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 [20]:

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

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 [10]. 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 datasets [6], and an updateable version of Naïve Bayes[6], which also works for smaller datasets. The current states of the relevance models are serialized and persisted after each incremental update. These models are ready-to-use 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*. 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. 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.

3.5 Making Use of Relevance Models

SMR only caters for learning and providing relevance models. This means that the actual usage of a model is up to the search engine owner. A relevance model RM takes a vector of interaction feature values $\boldsymbol{I}$ for a given query–result pair (q,r) and returns a corresponding relevance prediction $\widehat{rel}$, i.e., $\mathrm{RM}(q, r, \boldsymbol{I}) = \widehat{rel}(q, r)$.

This can, e.g., be integrated into a scheduled process of updating search result ranking according to a learning-to-rank function that contains $\widehat{rel}$ as a parameter (Fig. 2). The interaction feature values used for prediction could be those recorded by SMR and persisted by the Interaction Features Processor.

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. [20]) in terms of the

[6] `http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayesUpdateable.html` (2013-10-07).

prediction quality of the two systems. Second, we provide BBM (as a state-of-the art generative click model aiming at stream processing; cf. [16]) 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. 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. [10]).

4.1 Effectiveness

Method. Approximately 32 GB of raw tracking data were collected by SMR's interaction tracking facilities in May 2013 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. 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., *#conversions* $\ll$ *#search sessions*.

We divided the chosen raw interaction data into *10 distinct datasets 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 datasets with both systems yields a total of 20 datasets 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 [20], 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.

In analogy to [20], 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 suitable for cases with unbalanced classes [1].

Relevance models as provided by SMR and TMR are highly sensitive to layout specifics of the corresponding SERPs [20]. 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).

The Storm cluster used for evaluation was based on *Amazon EC2*[7]. 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[8].

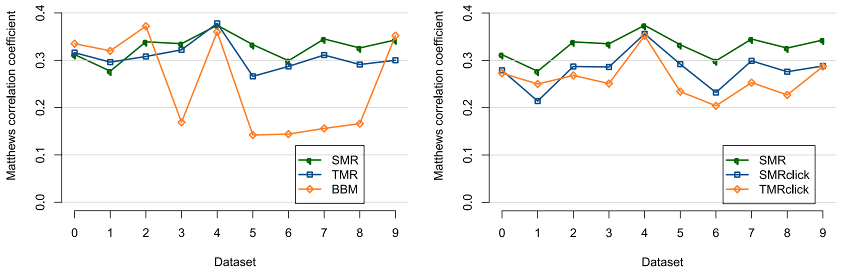

Fig. 3. MCC values for *DS0–DS9* (threshold = 0.5)

SMR vs. TMR. Based on the datasets described above, we trained a total of 20 Naïve Bayes classifiers (10 per system), as provided by TMR and SMR through the WEKA API. Thereby, our system used the updateable 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 [6]. All classifiers learned have been evaluated using *10-fold cross validation*, from which we obtained corresponding MCC values. As can be seen in Fig. 3, the difference between SMR and TMR is not significant across the 10 datasets. This result has been validated using a Wilcoxon rank sum test, with $p{>}0.05$ ($\alpha{=}0.05$, $W{=}75$, 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. 3 shows only MCC values at a

[7] `http://aws.amazon.com/ec2` (2013-09-30).
[8] `http://aws.amazon.com/en/ec2/instance-types/#instance-details` (2013-10-05).

threshold of 0.5, our result is underpinned by the exemplary receiver operating characteristic (ROC) curves depicted in Fig. 4, where SMR does not dominate TMR or vice versa. This is similar for the remaining nine datasets.

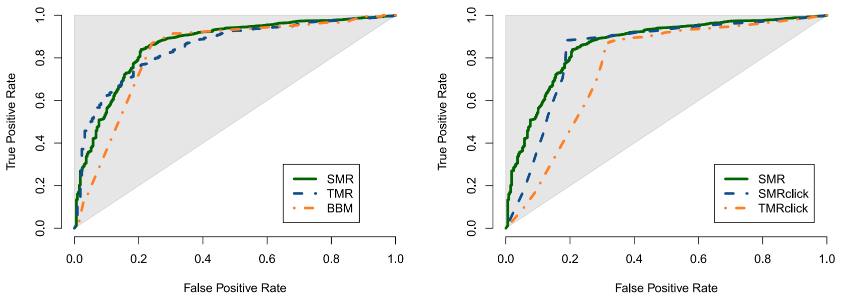

Fig. 4. ROC values for *DS7*

SMR vs. BBM. 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 [16]—and provided it with the exact same raw interaction logs. Fig. 3 shows that BBM yields slightly better predictions for four out of ten datasets (*DS0*–*DS2*, *DS9*) at a threshold of 0.5 while SMR has a better prediction quality for the remaining six datasets. 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. 4 suggests that predictions of BBM can be partly dominated by SMR's predictions for certain datasets. We expect SMR's prediction quality to increase with amounts of data larger than used in this evaluation. Thus, we hypothesize that our system can predict relevance at least as good as BBM, whose predictions are being successfully used in industry.

SMR vs. SMR$_{\text{click}}$ vs. TMR$_{\text{click}}$. 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$_{\text{click}}$ and TMR$_{\text{click}}$. Results show that SMR outperforms the click-only approaches across all 10 datasets (Fig. 3) based on *10-fold cross-validation*. Moreover, the MCC differences between SMR and SMR$_{\text{click}}$/TMR$_{\text{click}}$ are significant, as has been shown by two Wilcoxon rank sum tests (SMR$_{\text{click}}$: α=0.05, W=84.5, p<0.05, 95.67% conf. int. = [-0.075, -0.020]; TMR$_{\text{click}}$: α=0.05, W=90, p<0.01, 95.67% conf. int. = [-0.101, -0.044]). Our results are further supported by the ROC curves shown in Fig. 4, where SMR (area under ROC = 0.861) performs better than both SMR$_{\text{click}}$ (area under ROC = 0.834) and TMR$_{\text{click}}$ (area under ROC = 0.759).

These findings underpin that adding interaction data other than clicks yields considerable improvements for discriminative approaches, as has also been outlined in [11,13]. 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}).

4.2 Efficiency, Scalability and Robustness

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 [20]. 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 re-processed for an update, which yields a time-complexity of $O(q) + O(s)$ with s = #search sessions in new log, 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 re-processed 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, re-processing 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 re-process 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 depends on the complexity of the classifier used. For example, "constant time per example [i.e., a query–result pair in our case]" [6] if using a Hoeffding Tree. SMR needs ~2 hours for processing all search sessions in a 1.5 GB log using the cluster described in Section 4.1. 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 [16], 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 [5] cluster. BBM's time-complexity for updating a relevance model is $O(s)$.

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 1.

Robustness. Being based on Storm, SMR is a highly robust system by design. In particular, it features guaranteed message passing[9] and high fault-tolerance[10] 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 [16], 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 re-processed. In particular, this means that already computed values of interaction features are useless since contributions of already processed data can not 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 datasets 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.

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 dataset 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

[9] `https://github.com/nathanmarz/storm/wiki/Guaranteeing-message-processing` (2013-12-30).

[10] `https://github.com/nathanmarz/storm/wiki/Fault-tolerance` (2013-12-30).

data has lower quality. However, the different feature values for query–result pairs that appear early in a dataset 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 datasets, 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 datasets, 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 datasets 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 datasets. This and the fact that the datasets 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. This is, e.g., indicated in [20], where the authors work with datasets that are notably larger than 1.5 GB. Also, 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" [12].

How does SMR deal with click spam? Click spam is a major problem in systems where clicks are the main indicator for relevance [19]. 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 [19]. 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 [19] or the ranking framework described by [2].

Table 1. 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

Summary. Table 1 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.

5 Limitations and Future Work

The following section discusses limitations of SMR and provides an overview of potential future work.

As described, in this paper 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. Transferring SMR into such a more general context is our current work-in-progress.

Concerning the evaluation of our system, we had to rely on relatively small datasets 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 datasets that simulate a real-world setting of a timespan 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 [6].

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* [21]. 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" [21]. 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 [14].

6 Conclusions

This paper presented 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 re-processing 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 an analogous batch-wise approach 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 datasets. 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.

As future work, we plan to adjust SMR to more general settings besides travel search. Moreover, it is planned to further optimize the system regarding performance and perform an evaluation with even larger amounts of real-world interaction data.

Acknowledgments. We thank Christiane Lemke and Liliya Avdiyenko for supporting us with their implementation of BBM. This work has been supported by the ESF and the Free State of Saxony.

Gefördert aus Mitteln der Europäischen Union

Europa fördert Sachsen. ESF Europäischer Sozialfonds

References

1. Baldi, P., Brunak, S., Chauvin, Y., Andersen, C.A., Nielsen, H.: Assessing the accuracy of prediction algorithms for classification: an overview. Bioinformatics 16(5) (2000)

2. Bian, J., Liu, Y., Agichtein, E., Zha, H.: A Few Bad Votes Too Many? Towards Robust Ranking in Social Media. In: Proc. AIRWeb (2008)
3. Chapelle, O., Zhang, Y.: A Dynamic Bayesian Network Click Model for Web Search Ranking. In: Proc. WWW (2009)
4. Craswell, N., Zoeter, O., Tylor, M., Ramsey, B.: An Experimental Comparison of Click Position-Bias Models. In: Proc. WSDM (2008)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. CACM 51(1) (2008)
6. Domingos, P., Hulten, G.: Mining High-Speed Data Streams. In: Proc. KDD (2000)
7. Dupret, G.E., Piwowarski, B.: A User Browsing Model to Predict Search Engine Click Data from Past Observations. In: Proc. SIGIR (2008)
8. Guo, F., Liu, C., Wang, Y.M.: Efficient Multiple-Click Models in Web Search. In: Proc. WSDM (2009)
9. Guo, Q., Agichtein, E.: Beyond Dwell Time: Estimating Document Relevance from Cursor Movements and other Post-click Searcher Behavior. In: Proc. WWW (2012)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explor. Newsl. 11(1) (2009)
11. Huang, J.: On the Value of Page-Level Interactions in Web Search. In: HCIR Workshop (2011)
12. Huang, J., White, R.W., Buscher, G., Wang, K.: Improving Searcher Models Using Mouse Cursor Activity. In: Proc. SIGIR (2012)
13. Huang, J., White, R.W., Dumais, S.: No Clicks, No Problem: Using Cursor Movements to Understand and Improve Search. In: Proc. CHI (2011)
14. Hulten, G., Spencer, L., Domingos, P.: Mining Time-Changing Data Streams. In: Proc. KDD (2001)
15. Joachims, T.: Optimizing Search Engines using Clickthrough Data. In: Proc. KDD (2002)
16. Liu, C., Guo, F., Faloutsos, C.: BBM: Bayesian Browsing Model from Petabyte-scale Data. In: Proc. KDD (2009)
17. Marz, N.: Storm Wiki, `http://github.com/nathanmarz/storm/wiki`
18. Navalpakkam, V., Churchill, E.F.: Mouse Tracking: Measuring and Predicting Users' Experience of Web-based Content. In: Proc. CHI (2012)
19. Radlinski, F.: Addressing Malicious Noise in Clickthrough Data. In: LR4IR Workshop at SIGIR (2007)
20. Speicher, M., Both, A., Gaedke, M.: TellMyRelevance! Predicting the Relevance of Web Search Results from Cursor Interactions. In: Proc. CIKM (2013)
21. Tsymbal, A.: The problem of concept drift: definitions and related work. Technical Report, Trinity College Dublin (2004)
22. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, I.: Discretized streams: A fault-tolerant model for scalable stream processing. Technical Report, UC Berkeley (2012)