Query-Less: Predicting Task Repetition for NextGen Proactive Search and Recommendation Engines

[1]Yang Song, [2]Qi Guo

[1]Microsoft Research,
[2]Microsoft,
One Microsoft Way,
Redmond, WA 98052, USA
{yangsong, qiguo}@microsoft.com

ABSTRACT

Web search has been a *reactive* scenario for decades which often starts by users issuing queries. By studying the user behavior in search engine logs, we have discovered that many of the search *tasks* such as stock-price checking, news reading exhibit strong repeated patterns from day to day. In addition, users exhibit even stronger repetition on mobile devices. This provides us chances to perform *proactive* recommendations without user issuing queries. In this work, we aim at discovering and characterizing these types of tasks so that we can automatically predict *when* and *what types* of tasks will be repeated by the users in the future, through analyzing search logs from a commercial Web search engine and user interaction logs from a mobile App that offers proactive recommendations. We first introduce a set of novel features that can accurately capture task repetition. We then propose a novel deep learning framework that learns user preferences and makes automatic predictions. Our framework is capable of learning both user-independent global models as well as catering personalized models via model adaptation. The model we developed significantly outperforms other state-of-the-art predictive models by large margins. We also demonstrate the power of our model and features through an application to improve the recommendation quality of the mobile App. Results indicate a significant relevance improvement over the current production system.

Keywords

Web log analysis; User behavior; predicting future events

1. INTRODUCTION

Since the emergence of search engines in the 1990s, Web search has remained as a user-initiated paradigm for the past two decades. Essentially, it begins with a user first issuing a query to express her information need. Search engines then try to parse the query to understand user's underlying intent and provide a ranked list of results based on their relevance to the user query. Finally, the user elects to click some results and/or reformulates the query to repeat the process as needed. Modern Web search engines have evolved recently by better understanding user intent based on their search history, and therefore provide more customized results for individual users, which is often referred to as contextual search [13, 24] or personalized search [26, 21, 25, 20].

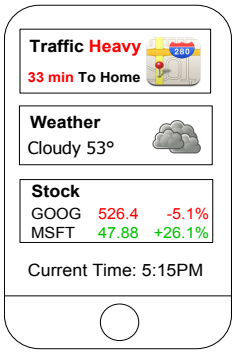
Nevertheless, as the world is becoming more mobile-centric, this old-fashioned query-driven search scenario and click-based evaluation mechanism can no longer catch up with the rapid evolution of user demand on mobile devices. On one hand, the limitation of smaller screen sizes, restricted typing area and pre-matured input methods has refrained users from expressing needs and thus discovering information as easily as they used to do on desktop computers [19]. On the other hand, users also demand more contextual information as mobile devices can grant easy access to the Internet and online services virtually anytime anywhere.

Therefore, a more user-friendly, mobile-centric and scenario-driven search paradigm that requires minimal user inputs is ready to come out. Recently, both Google and Microsoft have introduced their personal assistant tools on mobile clients, namely Google Now and Microsoft Cortana. Both tools provide proactive services and recommendations such as stock quotes and nearby restaurants. Figure 1 illustrates a mocked example of such tool where three services (weather, stock and traffic) are shown, in which, the tool learns from past user behavior that the user might get off work soon and head home given the current context (time, location), and thus ranks the traffic information on top to help navigation.

In this work, we aim at learning user preferences and interests based on their past search history. Specifically, we focus on extracting user *search tasks* that have *regular* repeated patterns and predicting when users will perform those tasks again in the future. One of the immediate application of our model is to improve the triggering and ranking of the proactive personal assistant. However, the significance of our work is far beyond that. For example, the learnt model can be directly applied to improve search personalization, guide the caching and pre-fetching services for search engine backends, or design better query classifiers for tail queries.

More formally defined, we address the problem of *predicting task repetition* in this paper, that is:

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883020 .



User	Time	Activity	Service
U_1	07/01/14 6:30am	View	Weather
U_1	07/01/14 8:55am	Click	Stock
U_1	07/01/14 5:05pm	Click	News
.....			
U_2	07/03/14 10:30am	View	Stock
U_2	07/04/14 10:45am	View	Stock
U_2	07/05/14 10:14am	View	Stock

Figure 1: A simple demo showing the interface and some functionalities of the mobile personal assistant.

Given a user's past search history, organized as search tasks, determine *which* tasks will be repeated during *what* hour of the day (e.g, whether a user will perform a *stock-related* task between *5pm and 6pm today*).

Previous research has studied Web page re-visitation [10, 1, 2, 4] and recurrent event queries [29] and demonstrated that repeated patterns indeed exist in both Web page visits and user queries. Those work often examine the patterns by aggregating data across different users and domains, which makes the discovery difficult to adapt to individual users. In comparison, our work differs significantly from all previous efforts in that we target at *individual users*, where the user-level data is much *sparser* and thus poses greater challenges to the methodologies.

To be concrete, we make the following contributions:

- We leverage a structured learning framework to extract long-term user search tasks from logs. These search tasks allow us to efficiently identify user repeated search patterns (Section 4.1).
- We propose a novel information-theoretic approach to determine the *predictability* of user tasks, i.e., whether a task represents one-time information need or exhibits recurrent patterns. This helps filtering out non-repeatable tasks and allows training models with less noisy data (Section 4.2).
- We introduce a set of novel features to characterize user behaviors and task repetition patterns for this new problem (Section 4.3).
- We introduce a deep learning model for prediction. We propose to train global prediction models and then adapt to individual users using model adaptation (Section 4.4, 4.5).
- We perform extensive empirical evaluation using real-world search engine logs to demonstrate the superiority of our DNN models over the state-of-the-art models in predicting the task repetition, and show that our novel features and models can help significantly improve the ranking of proactive recommendations (Section 5).

2. RELATED WORK

In this section, we briefly review previous research on (1) user re-visitation patterns, (2) search task continuation, and (3) personalized search.

Adar et al. [1] analyzed Web page re-visitation patterns from a large collection of Web logs. The authors discovered the relationship between re-visitation frequency and the amount of changes of different pages which showed a positive correlation. Furthermore, the authors provided a deeper analysis on four different re-visitation patterns [2], namely fast, medium, slow and hybrid revisits. The study indicated that Web pages can be clustered based on the user re-visitation patterns using a hierarchical clustering method. From the perspective of user queries, Zhang et al. [29] proposed to learn recurrent event queries from search engine logs. The authors first identified repeated user events by different time intervals from weekly to bi-annually. They then proposed a predictive model using a set of query-based and click-based features. They compared the performance of several machine learning methods and showed that the ranking of recurrent queries can be improved substantially.

Long-term search task segmentation has recently emerged as a hot topic for user modeling. Wang et al. [22] proposed a structural learning algorithm to find the best links among user queries. Gupta et al. [7] developed a binary classifier to determine whether two queries belong to the same information need. Lucchese et al. [16] constructed task-to-task transition graph to make predictions for new queries. Similarly, the authors in [9] built a task graph to link similar tasks to recommend related tasks in the exploratory search scenarios. On the other hand, task continuation aims at predicting future events. Wang et al. [23] tried to predict search task continuation when the users switch devices from PC to tablet using binary classifiers. The most related work to ours is from [3], where Agichtein et al. proposed to predict whether a searcher will continue an unfinished task within the next few days. The authors introduced a feature-based predictive model that was able to outperform human predictions. Comparatively, task segmentation can be formulated as $P(T = T_i|Q_{new})$ where a new given query Q_{new} is to be linked with previous tasks T_i. Task continuation prediction is instead to predict $P(T = T_i)$ without any given queries. Our work is much more challenging by adding the time requirement to the prediction, i.e., $P(T = T_i|hour)$, but also makes the results more useful in practical applications.

Personalization is at the core of proactive recommendations. In literature, personalized search is the most related area for this aspect, to which much research work has been devoted. For example, White et al. [25] addressed the issue of identifying valuable user groups, namely *cohorts* by implicitly modeling user search tasks and discovering users who often issue similar tasks. The users were then grouped based on their task similarity so that other users' search behavior can be leveraged to enhance the relevance for the current user. Yan et al. [26] extended the previous work by constructing both predefined and dynamic cohorts from user behavioral data. They proposed to model query acronyms so that new, unseen queries from a user can also be modeled. From a modeling perspective, Wang et al. [21] proposed a model adaptation framework to suit different users' information need. The authors claimed that users have different bias when viewing search results. e.g., some users prefer well-known domains, while others focus more on title match. They introduced operations to transform the feature matrix according to individual users search history. The authors in [20] also addressed the modeling issue by introducing deep learning for model adaptation. The authors argued that by performing continue-train on global models using user-specific data, it can easily adapt a user-independent model

to different user preferences which lead to better relevance than previously proposed methods.

3. DATA DESCRIPTION

There are two types of data that we use in this paper: (1) Web search logs, and (2) mobile App usage logs.

We collected a large sample of user search logs from a commercial Web search engine. The timespan of the logs is approximately three months (May 1st to July 31st, 2014). We selected users within the US search market and filtered non-English queries. The search logs contain user interactions with the search engine, including a user's anonymized identifier (UID), the event timestamp, raw queries the user issued, URLs the user clicked on as well as some geographical information about the user. These events were grouped into sessions where each session contains one or more events. A session ends when a 30-minute inactivity period is detected. A sample of search logs can be seen in Table 1. Overall, we sampled roughly 1 million users with 80 million sessions.

We also collected a sample of user logs from a personal assistant App on a major mobile platform. The App makes proactive recommendations to users. The recommended services include weather, stock, traffic, places nearby and so on. To align with the Web data, we also collected three-month App logs during the same time period. Specifically, each entry of the App logs contains the user interaction with the App, which contains the user's anonymized ID, the event timestamp, the name of the services the user interacted with, the type of interaction, and some additional information such as the link of the URL for News. For the type of interactions, we recorded several activities including user clicks and the dwelltime on the recommendation. This is because services like stock and weather often receive much less clicks than others. Instead, users may acquire sufficient information by simply viewing the recommendation. Thus, the dwell time information provides important implicit relevance feedback [8]. To calculate the dwell time on viewing a recommendation on the proactive impression, we insert Javascript to each impression, which records the viewport changing events as well as the positions and sizes of the recommendations in the impression, and sends the buffered events as HTTP requests to our server. The right part of Figure 1 shows examples of user activities. In total, we collected 60K user sessions from 50K sampled users.

4. METHODOLOGY

This section introduces our methods. We first review the algorithm used for extracting user tasks. We then propose a way to determine the predictability of the tasks. After that, we introduce the features and learning method for prediction and discuss how we adapt it for individual users.

4.1 Get long-term user tasks

Since search engine logs are often ordered based on the event timestamp, previous research has shown the drawbacks of such ordering that can cause suboptimal understanding of user's underlying search intent [14, 15]. In practice, user's information need indeed spans multiple search sessions, or even multiple days which may contain many search queries and result clicks. Table 1 shows an example of search logs that contains 9 search sessions in three days from the same user. It is quite evident that there exists three major tasks within these sessions: (1) checking the stock price of *tsla*, (2) navigating to facebook.com, and (3) playing online games. Note that all three tasks spaned multiple sessions.

Time	Query	SessionID	TaskID
07/01/2014 08:07:05	tsla	1	1
07/01/2014 19:09:33	facebook	2	2
07/01/2014 19:33:04	free games	2	3
07/01/2014 23:11:25	facebook	3	2
07/02/2014 08:33:25	tsla	4	1
07/02/2014 13:22:17	tsla	5	1
07/02/2014 19:06:15	free online games	6	3
07/03/2014 08:55:53	tsla	7	1
07/03/2014 15:56:19	facebook com	8	2
07/03/2014 16:23:22	facebook	8	2
07/03/2014 19:21:05	free games	9	3

Table 1: An example of user search sessions and the tasks segmented by the learning algorithm. For space concern, the clicked URLs are omitted here.

Therefore, in order to extract the underlying user search tasks from logs, we propose to leverage a structured learning framework [22] to find the optimal hidden structure within user sessions. Specifically, the authors in [22] introduced a supervised learning framework that makes use of the bestlink support vector machines (SVM) to minimize the error between the predicted task partition and the ground-truth partition, as well as maximizing the margin between them. Thus, given a set of query logs with ground-truth, $\{Q_n, y_n\}_{n=1}^N$, where each query sequence $Q_n = \{q_{n1}, ..., q_{nm}\}$ corresponds to a user u_n's history and y_n are the annotated task IDs, the optimization framework is specified as:

$$\min_{w,\xi} \ \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i^2 \qquad (1)$$
$$s.t.\ \forall i, \quad \max_{h\in\mathcal{H}} w^{\mathsf{T}}\Phi(\mathcal{Q}_i, y_i, h) \geq$$
$$\max_{(\hat{y},\hat{h})\in\mathcal{Y}\times\mathcal{H}} [w^{\mathsf{T}}\Phi(\mathcal{Q}_i, \hat{y}, \hat{h}) + \Delta(y_i, \hat{y}, \hat{h})] - \xi_i$$

where $\Delta(y_i, \hat{y}, \hat{h})$ specifies the error between the prediction $\hat{y}$ (assigned by $\hat{h}$ the latent structure) and the ground-truth y_n, $\{\xi_i\}_{i=1}^n$ a set of slack variables for relaxation in training and C the balancing factor. $\Phi(\mathcal{Q}, y, h)$ is a set of feature vectors. The above equation is solved by an iterative algorithm that constructs a sequence of convex problems during each iteration so that the objective function is guaranteed to decrease [5].

The features used in this model include the cosine similarity between query terms, the edit distance between two query strings, the domain similarity between the clicked URLs and etc. In total, a set of 26 features are used.

4.2 Predictability of Task Repetition

Once the user tasks have been segmented, the next step is to determine whether a specific task has regular repeated patterns so that its next occurrence can be reasonably predicted. Some tasks are in nature more regular in repetition (e.g., stock-price checking) while some others are more arbitrary (e.g., social network site checking) or rarely re-occur (e.g., hotel booking) on a daily basis. Thus, filtering those non-repetitive tasks out can ensure a more consistent and less noisy set of data for model training. Note that some task repetition patterns may share across different users while

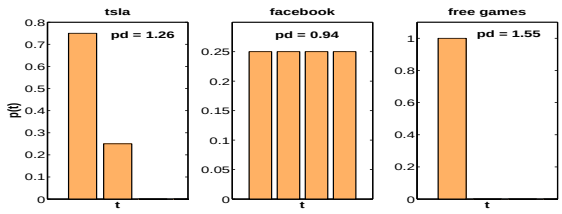


Figure 2: The probability distribution of the three tasks and their predictability scores pd with $\lambda = 0.5$.

some other task repetition patterns are more personalized. Again, take Table 1 for example, it is evident that task 1 and task 3 exhibit regular repetitions: the user routinely checks stock prices around 8am in the morning (with only one exception) and plays online game during 7pm in the night. On the other hand, task 2, the navigation to the social network site, is less regular and predictable for the user, as she tends to issue the query at arbitrary times of the day. Since our objective is to perform hourly predictions of the occurrences of daily routines, we propose an information-theoretic metric to measure the predictability of user tasks. Given a task T_{nm} from user u_n, the probability $p(t)$ that the user issues a query at hour t is defined as the normalized score of the query count during hour t, $|T^t_{nm}|$, divided by the total query count in that task $|T_{nm}|$. The predictability of the task is then specified as

$$pd(T_{nm}) = \exp\left(\sum_t p(t)\log p(t)\right) + \lambda \log\left(|T_{nm}|\right), \quad p(t) = \frac{|T^t_{nm}|}{|T_{nm}|}, t \in [0, 23], \lambda \geq 0 \tag{2}$$

where the first part inside $\exp(\cdot)$ is essentially the negative entropy of the probability distribution, while the second part regularizes the length of the task. This equation reflects our intuition that, longer tasks with lower entropies are generally more predictable since they often show consistent repetition patterns. As we show later in Section 5.1, these two factors are indeed good predictors of predictability of task competition, and combining the two results in better correlation with predictability and helps remove data noise and improve model accuracy.

Figure 2 shows the probability distribution in bar plots for the three tasks in Table 1. We also show their predictability scores by setting λ to 0.5. It can be observed that the *game* task, which has clear repetitive patterns, has the highest score (1.55). On the other hand, the *facebook* task that exhibits uniform probability distribution, gets the lowest score (0.94). Later, we will show that the optimal value of λ is set by using cross validation on a held-out data set.

4.3 Feature Generation

To accurately capture the characteristics of task repetition patterns, we introduce a set of features constructed from user's behavioral histories. The high-level description of the features is listed in Table 2. Compared to features proposed in similar work [29], our features are more focused on individual tasks and users and thus can capture the characteristics of repetition better for personalization.

(**Query-level Features**) query-level features include several time-based features like the day of the week, hour of the day etc. These categorical features are transformed into binary features using corresponding number of bins. Three click-based features are also included to identify user activities during the past few days/months. Notice that the particularly useful clicks for prediction are those same-hour clicks in user click history. That is, to predict whether a user will issue a *stock* query between 8am and 9am for a given day, a useful feature would be the total number of clicks of stock queries between 8am and 9am of the same user during the past few days. To determine the category of a query, we built a set of binary classifiers using human-labeled data for the following 12 categories: *news, shopping, stock, weather, travel, local, food, sports, games, movie, tech.*

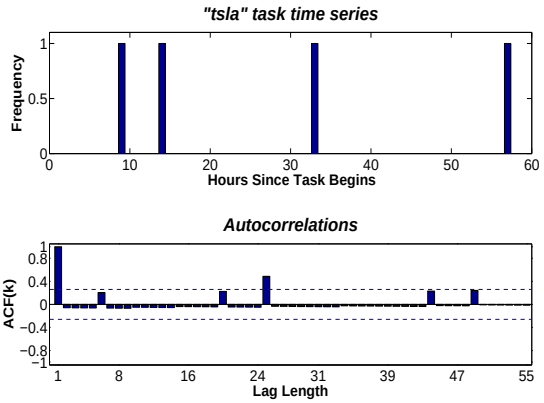


Figure 3: An illustration of a task time series and its corresponding autocorrelation function. The two dashed lines show the rejection region for $\alpha = 0.05$.

(**Task-level Features**) task-level features include the length of the task that indicates how likely the task is repeated by the user. The category of a task is assigned using majority voting of all queries in the task. In addition, to measure the task repetition frequencies at daily or longer period of intervals, we apply autocorrelation to calculate the intervals:

$$ACF(k) = \frac{\sum_{t=1}^{T-k}(y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2}, k \in [1, T) \tag{3}$$

where $y_t = |T^t_{nm}|$ is the query count at the t^{th} hour, T the total number of hours the task spans, $\bar{y}$ the mean of the time series. The denominator of eq. (3) is the unscaled variance of the data. $ACF(k)$ measures the autocorrelation of a task at lag k. Figure 3 shows the example for the previous *tsla* task. The top figure plots the time series of the task where each bar indicates a query issued during that hour. The bottom figure shows the autocorrelation function from lag 1 to 55 hours. We can clearly observe a peak at lag 24 (lag at 1 always equal to 1) which is exactly one-day apart, showing a strong *daily* repetition pattern.

(**User-level Features**) user-level features measure how actively the users have been using the services in the recent days. These features include total query count, same-hour total query count, as well as the number of active days (days that a user issued at least one query). In addition, user activity during the immediate past hour is also considered as a feature due to its time proximity.

Notice that the above description of features measure user activities based on their query history, which is primarily for the search logs. For the mobile App logs, since users do not

issue any search queries, the measurement of user activities is replaced by the name of the services that users interacted with (e.g., view, swipe, click etc.). To be concrete, the previous mentioned category of queries is replaced by a set of services, namely *local, weather, traffic, finance, sports, map, news* and *travel*. Overall, a set of 96 binary and numerical features are generated for each training instance.

4.4 Task Prediction via Classification

Although it seems tempting to predict task repetition using time-series prediction models given the temporal nature of the data, it turns out to be sub-optimal to do so due to discontinuity of the repetitions. In other words, there are excessive zeros in the temporal activation function of a task in the data. Take Figure 3 for example, the task *tsla* is only active at 4 timestamps with 56 zeroes in the 60-hour time window. Due to such level of sparsity, even the most sophisticated time-series models are incapable of making reliable predictions, as we shall see in the experiment section.

Consequently, we formulate the prediction problem as a binary classification task. Specifically, a deep neural network (DNN) is used to train a probabilistic classifier with the features defined in Table 2. DNN has emerged to be a powerful tool for scalable learning tasks such as personalized search [21, 20]. It is also a natural choice for performing adaptation as we shall see in the next section. Given an input feature vector x_i with its label y_i, the neural network passes the input into a set of hidden neurons by performing a weighted sum of the inputs using a non-linear activation function a, which is usually a sigmoid function $a(x) = \frac{1}{1+e^{(-x)}}$, and the output $\hat{y}_i$ is calculated as $\hat{y}_i = \sum_j a\left(\sum_k a(w_k x_i + b_k) + b_j\right)$, where w_k is the weight vector for the kth hidden neuron and b_k the bias term. The network updates the weights by calculating the error between the prediction $\hat{y}_i$ and the true label y_i and back-propagate the errors to each neuron using stochastic gradient descent.

For training, each query within a task is treated as a positive instance (+1). To generate negative training examples (-1), we randomly select hours between the first and last event within the task where the user did not issue any queries. Take the facebook task for example, as illustrated in Table 3, this task consists of four positive examples from four different hours. For simplicity, we use an index to track the time of each instance where 07/01/2014 00 : 00 : 00 has index of 0. The index increases by 1 for each hour. The largest index for the task is then 64. Then, four negative examples are generated by randomly choosing four indices between 19 and 64. To alleviate the class imbalance issue, we generate roughly the same number of negative examples as the number of positive examples for training, while keep the test data with the original imbalanced class distribution.

To train a global model for prediction, we remove the user IDs by aggregating all positive and negative training data into one training set. This essentially ignores the user preferences among tasks by considering each type of task as a single classification problem. In the next section, we show how to address this issue.

4.5 Model Adaptation for Personalization

A global model is essential to capture the overall characteristics for different types of tasks. However, users may exhibit distinct behaviors that cannot be accurately captured by a user-independent global model. For example, while most of the users tend to check stock prices during normal financial operation hours, one particular user may elect to perform such action during midnight every day. In machine learning, such scenario is often referred to as model mismatch, which often happens when a model is trained on a source domain while the target domain exhibits different feature distribution. A common approach to address this issue is to apply model adaptation from the source domain(s) to the target domain(s). In the information retrieval community, researchers have recently discovered that by performing continue-train on a DNN model to user-specific data [20, 21], the adapted models perform significantly better than the global model for most users. Adaptation to individual users have two advantages over re-training a global model: first, it is much more computationally efficient to perform continue-train since the number of user-specific training points tends to be very limited compare to the user-independent data; in addition, continue-train puts more emphasis on user data so that the user preference is not overwhelmed by the much larger size of global training data.

Time	Index	Query	Label
07/01/2014 19:09:33	19	facebook	+1
07/01/2014 21:00:00	21	<no query>	-1
07/01/2014 23:11:25	23	facebook	+1
07/02/2014 05:00:00	29	<no query>	-1
07/02/2014 11:00:00	35	<no query>	-1
07/02/2014 18:00:00	42	<no query>	-1
07/03/2014 15:56:19	63	facebook com	+1
07/03/2014 16:23:22	64	facebook	+1

Table 3: An example of the training data generation for the facebook task. The four negative examples are randomly picked hours within the task events.

In this paper, we perform two types of adaptations. The first one is within-domain adaptation, which adapts a globally-trained model from search logs to individual user's search tasks. Due to the extremely large size of the search logs, it is infeasible to train a global model based on all labeled data. Instead, a small sample of randomly selected training data is more preferable to train a model in reasonable time. The model can then be adapted to user-specific tasks by training models for each task of a targeted user.

The second type of adaptation is cross-domain adaptation, which adapts the global model from search logs to mobile App logs. Compared to search logs, the size of App logs is much smaller and thus creates challenges in modeling individual user's preferences. Instead, we have discovered strong correlation between the usage of Web search and mobile App [18], so that adapting from the model trained on Web log can alleviate the sparcity issue.

4.5.1 Mining Cohort Behavior

To further address the sparseness issue in training data, we propose to enrich the user training data by leveraging behaviors from similar users (cohorts). The underlying assumption is straightforward: similar users often perform similar activities at similar time. It has been shown that leveraging cohort behavior can significantly improve the quality of personalization [25]. In this work, we leverage both task-based features and behavior features to discover cohorts. Specifically, the task-based features are binary indicator variables where 1 indicates a user has performed a particular task during hour h of the day ($h \in [0, 23]$). For behavior features, we include users' geo-location (at State level), types of Web

Type	Feature	Description
Query-level	DAY-OF-WEEK	the day of the week when the query was issued, between [0, 6]
	IS-WEEKDAY	whether the query was issued during weekday (1) or weekend (0)
	HOUR-OF-DAY	the hour of the day when the query was issued, between [0, 23]
	TIME-OF-DAY	the time period of the day: morning (0), afternoon (1), evening (2) or night (3)
	IS-WORKHOUR	whether the query was issued during work hours between 9am and 5pm
	TOTAL-CLICK-1	total clicked URLs in the previous day during the same hour
	TOTAL-CLICK-7	total clicked URLs in the past week during the same hour
	TOTAL-CLICK-30/90	total clicked URLs in the past 1(3) months during the same hour
	QUERY-CATEGORY	category of the query (e.g., finance, weather)
Task-level	TASK-LENGTH	total number of queries in the task
	TASK-AC	the autocorrelation function of the task
	TASK-CATEGORY	category of the task (see text for details)
User-level	USER-QCNT-1/3/7	total number of queries issued by the user during the last 1/3/7 day(s)
	USER-QCNT-HOUR-3/7	total number of queries issued during the last 3/7 days at the same hour
	USER-ACTIVE-DAY-3/7	total number of active days in the past 3/7 days
	USER-LAST-HOUR-QCNT	total number of queries issued during last hour

Table 2: List of features used for training classification models.

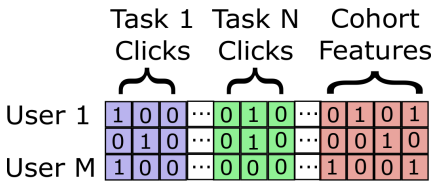


Figure 4: The user-behavior matrix used to discover similar users (cohorts).

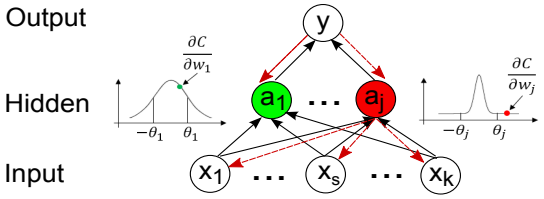


Figure 5: The illustration of truncated gradient for a single hidden-layer DNN.

browsers (IE, Chrome, etc.) and search entry points (front page, MSN, toolbar, etc.). We further transform these categorical features into binary features.

The resulting binary-valued matrix $M \in R^{m \times n}$ is a sparse matrix with around 600 columns as shown in Figure 4. We perform dimension reduction using SVD to choose k_1 hidden dimensions ($M = U\Sigma V^T, \tilde{M} = U\Sigma^{[1:k_1]}V^T$), and then use the k-means algorithm on matrix $\tilde{M}$ to find k_2 user clusters. Within each cluster, majority vote is performed for each task to fill in the zero values, i.e., determine whether a user will issue the task during a certain hour. Those entries that are predicted to be 1 are used as enriched positive training examples during model adaptation, which we discussed previously. In our experiment, k_1 and k_2 are set as 20 and 200 empirically using cross validation.

4.5.2 *Truncated Gradient to Prevent Model Over-fitting*

It has been shown that performing continue-train on neural network can lead to model over-fitting if the parameters are not well-tuned[20]. Such an issue often happens in online learning where the weights are sparse due to the large number of parameters and features. Traditional approaches to tackle this issue including Lasso (L_1 regularization) often failed to work in an online setting. Recently, researchers have found that using truncated gradient to update the weights tends to work very well for deep neural networks [12]. Specifically, a function T with parameter θ is used to control the weight updates during back propagation for the k^{th} neuron, $w_k \rightarrow w_k - \eta T\left(\frac{\partial C}{\partial w_k}, a(k), \theta\right)$, where $a(k)$ is the output from the sigmoid, C the cost function. The gradient is truncated when the output is within the range of $[-\theta, \theta]$ according to T.

Specifically, each neuron has its own θ_k which is often estimated from a held-out validation set and thus having different truncation criteria. Figure 5 presents an example of how truncated gradient works. In this example, the output layer sends back the error to the hidden layer. The first neuron is truncated because the gradient is within $[-\theta_1, \theta_1]$ and therefore no further back propagation is performed to the input layer. On the other hand, the j^{th} neuron continues to propagate the error back to the input layer since $\frac{\partial C}{\partial w_j}$ is greater than θ_j.

5. EXPERIMENT

In this section, we present empirical study on the methods proposed in this paper. Overall, we perform two sets of experiments to assess our methods: (1) predicting Web search and mobile App task repetition and (2) improving the ranking of mobile proactive recommendations.

5.1 Experimental Data Preparation

The evaluation objective for prediction experiments is that given the first two-month data for training (from May 1st to June 30th, 2014), we want to assess the predictive quality for individual user's tasks issued during the third month (July 1st to July 31st, 2014). First, we use the technique proposed in Section 4.2 to filter out non-repeatable tasks. To do so, we first determine the optimal value of λ in eq.(2). We asked a human judger to manually label 100 randomly selected tasks from the training data as either *predictable* or *non-predictable*. Table 4 shows the statistics. It is evident that both the length and entropy exhibit different distributions for the two classes. The $p-val$ from the t-test confirms that

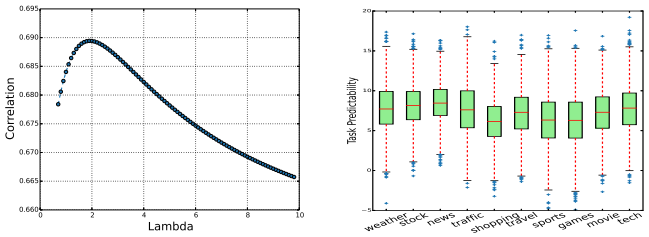
Table 4: **Length and entropy distribution (mean ± std) for 100 manually labeled instances.**

	Predictable (+1)	Non-Predictable (-1)	p-val
Length	32.02 ± 58.65	18.4 ± 24.02	0.017
Entropy	1.36 ± 0.54	1.97 ±0.69	0

Table 5: Statistics of experimental data sets.

	# Users	# Tasks	# Train Instances
Prediction Set	184,219	368,914	1,026,213
Ranking Set	12,131	113,369	49,456

the differences are statistically significant. We then determine the value of λ by maximizing the correlation between the human labels and the predictability scores, which turns out to be around 2.0 as shown in Figure 6(a). Comparatively, the correlation between label and task length (0.59), label and entropy (0.65) are much lower, showing the effectiveness of combining two variables. Consequently, we filter out the bottom 5% non-predictable tasks, which roughly corresponds to tasks with predictability score ≤ 5. We further divide the predictability into tasks categories as shown in Figure 6(b), where we observe that *news* and *stock* tasks have the highest scores among all types of tasks.



(a) (b)

Figure 6: Predictability breakdown. (a) Correlation between λ and human labels. Optimal correlation achieved at $\lambda = 2$. (b) Predictability by task categories. News & stock tasks have the highest scores.

After filtering, we split both data sets into two halves based on timestamps, where first half in each data set is used to train user-independent global models. We further split the second half into halves based on timestamps for adaptation and test, respectively, so that the obtained 25% user-specific data can be used to perform model adaptation on top of the trained global model using the first half of the data. Figure 7 illustrates the data splitting process.

For the second experiment, the goal is to improve the ranking of proactive recommendations. Specifically, we aim at improving the NDCG score of the ranked services by introducing our models and features into the existing production ranker. To do so, we randomly sample a subset of users from the entire App user base between August 2nd and 15th, 2014, where the data from August 2nd to 10th is used for training and validation, and the rest for testing purpose. Table 5 lists the statistics of the experimental data.

5.2 Methods Compared

We compare with several baseline systems:

• **Always Majority**: always predicts the majority class, which is the negative class in our problem, i.e., the task does not repeat at any point in time.

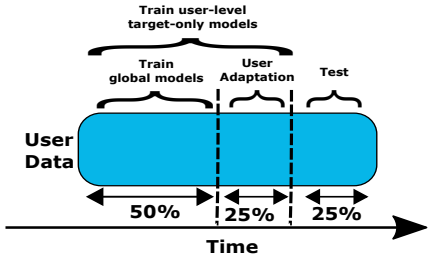


Figure 7: This figure shows the data splitting process for training, adaptation and testing.

• **Always Positive**: always predicts the positive class, i.e., the task repeats at every point in time. This is to complement the majority baseline.

• **Maximum Likelihood Estimation (MLE)**: this approach estimates task repetition with only user previous history. Specifically, it estimates the most likely time based on the summation of user previous queries during each hour, and choose the hours with the most queries issued to be the predicted hour.

• **Zero-inflated Poisson Autoregression (ZIM)**: the ZIM model [28] is introduced to model time-series data that contains excess zeros, which is applicable to our situation. The model is extended from the zero-inflated Poisson (ZIP) distribution, with probability mass function defined as

$$F_{Y_t}(y_t|D_{t-1}) = w_t I(y_t = 0) + (1 - w_t)\exp(-\lambda_t)\lambda_t^{y_t}/y_t!,$$

so that when $y_t = 0$ (the count is zero), $F_{Y_t}(y_t|D_{t-1}) = w_t + (1 - w_t)\exp(-\lambda_t)$ where zeros are generated from a Bernoulli distribution. On the other hand, the counts data are generated from a Poisson distribution when $y_t > 0$, with $F_{Y_t}(y_t|D_{t-1}) = (1 - w_t)\exp(-\lambda_t)\lambda_t^{y_t}/y_t!$. In our experiment, we use the standard R package[1].

• **Recurrent Neural Networks (RNN)**: RNNs have shown to perform well on time-series data prediction [6]. Among them, one type of RNNs is referred to as Echo State Network (ESN) where a large set of dynamical reservoir is randomly constructed from many neurons with fixed structure and input weights. The model is efficient in training since the back-propagation only needs to be performed between the output layer and the reservoir layer. In order for ESN to work, the reservoir needs to satisfy the echo state property where the spectral radius of the reservoir weight matrix must be less than unity. ESN has shown better predictive power on time-series data like stock data [17] than similar models (e.g., Kalman Filters).

• **Adaptive Support Vector Machines (aSVM)**: SVM has shown to be a powerful tool for classification. The adaptive SVM method works by adapting one or multiple existing SVM models to a new domain, which has shown successful applications on both classification [27] and ranking [21]. The underlying assumption is that the original domain shares some similarity with the target domain, while the target domain exhibits certain characteristics different from the original. Due to the data sparsity of the target domain, adaptation often outperforms the model trained only on the data from target domain. We use the aSVM package[2].

[1]http://cran.r-project.org/web/packages/ZIM/index.html
[2]http://www.cs.cmu.edu/∼juny/AdaptSVM/index.html

Table 6: Comparison of predictive performance for the five models. *-G = Global Only models. *-T = Target Only models. *-A = Adaptation models. ** = statistical significance with $p-val < 0.01$. †A global model using 75% of training data is also trained to show fair comparison to adaptation models.

	Web-Web		Web-App	
Models	Precision	Recall	Precision	Recall
Always Majority	0.0000	0.0000	0.0000	0.0000
Always Positive	0.0403	1.0000	0.0105	1.0000
MLE-G	0.3146	0.3507	0.2853	0.3085
ZIM-G	0.2957	0.3387	0.2745	0.3272
RNN-G	0.3046	0.3365	0.2659	0.2912
aSVM-G	0.4621	0.6082	0.4511	0.6065
DNN-G	0.5164	0.6805	0.4271	0.6176
†(75%) DNN-G	0.5377	0.6811	0.4695	0.6199
MLE-T	0.3397	0.3647	0.2856	0.2676
ZIM-T	0.2231	0.2584	0.2197	0.2099
RNN-T	0.2587	0.2636	0.252	0.2576
aSVM-T	0.4291	0.4463	0.4046	0.3739
DNN-T	0.4595	0.4698	0.4319	0.3767
DNN-T(c)	0.4587	0.4922	0.4285	0.3972
aSVM-A	0.5725	0.6831	0.5035	0.6034
DNN-A	**0.6163****	0.7265	**0.561****	0.6469
DNN-A(c)	0.6151	**0.7489****	0.557	**0.6596****

5.3 Predictive Performance

We present the experiment setting and results in the following section.

5.3.1 Parameter Settings

Before proceeding, we briefly discuss the parameter settings in the experiments. We tested various architectures for both neural networks using 10% hold-out data. Due to the large amount of experiments, we fixed the training iterations to be 2000 without performing early stopping. For the baseline RNN method, we found that with a reservoir size of 200 and an additional 10-node hidden layer between reservoir and the output, the model tends to perform the best. For our DNN method, the best performed model includes a hidden layer of 20 neurons with a fixed learning rate of 0.01 and a 0.1 momentum. For the aSVM model, RBF kernel with default parameters performs the best.

Note that among all methods compared, only aSVM and our DNN approach can perform model adaptation. Therefore, for all other methods, we train two models for each of them: *Global Only*: for each type of tasks, aggregate data across all users and train a single model; and *Target Only*: for each user, train individual models for each tasks using only that user's data.

5.3.2 Overall Performance Comparison

We use the precision and recall for the positive class as our metrics. Table 6 shows the overall predictive performance of the five models in terms of precision and recall. The Majority baseline (always predict -1) gets 0 for precision and recall for obvious reason. The Always Positive baseline reaches 1.0 for recall but very poor precision due to the imbalance of the test data set. The best performance is achieved by our DNN model with adaptation (DNN-A), followed by the aSVM model. The baseline RNN method which treats the data as time-series performs relatively poor compared with our model. The same observation can be found for the zero-inflated model ZIM, where the model performs worse than MLE. Although ZIM is designed to handle data with excessive zeros, previous studies have primarily focused on data such that the number of zero entries is typically no more than 50%. While our data, expressed as time-series, usually consists of 80% or more zero entries. These results have confirmed our assumption that time-series predictions do not suit our data well, but instead classification-based approach should be employed.

On the other hand, it can be seen that performing adaptation significantly boosted the performance from the global-only models, which in turn outperforms the target-only models. This gives us three implications. First, the same types of tasks indeed bear common repeated patterns across different users, which is why global models can capture most of their characteristics. Second, using only user's own data to train individual models does not perform as well as we expected, due to the sparcity of user-level data. Finally, users' unique characteristics that cannot be captured by global models can be better-explained after performing model adaptation.

By enhancing user-specific training data with cohorts (DNN-T(c) and DNN-A(c)), we notice that recall can be further improved while precision gets some negative impact. This indicates that more over-triggering can happen with artificial adaptation data. Whether or not this benefits the users depends heavily on the end-to-end scenarios. For example, in our later experiment of proactive recommendation, over-triggering seems to work better than under-triggering.

Note that global-only models use 50% data for training while the adaptation models benefit from additional 25% user-specific data, which may cause bias in performance comparison. To remove such bias, we re-trained a best-performing global-only model, namely DNN-G, using both parts of data (so now 75% training data) and evaluate on the same 25% test data, as shown in Table 6. We can observe that by using more data, the global DNN model indeed improves its own performance by about 4%. Nevertheless, we can still observe significant gap between the global models and the adaptation models even with the same amount of training data.

5.3.3 Performance Breakdown

Comparatively, Web-Web adaptation gives more promising results than Web-App adaptation. The DNN-A model shows 5% relative difference in precision. After deeper analysis of the data, two reasons are identified. The first reason that causes the discrepancy is due to the amount of adaptation data. In general, we have much more user-level data from Web search than mobile App. This creates certain difficulty in model adaptation that sometimes causes overfitting during adaptation. In addition, we discovered that several types of tasks demonstrate different patterns between Web and App, which causes the adaptation to be inefficient. Thus, it is important to breakdown the performance by task types for further analysis.

Table 7 shows the model performance for the top-4 task types (weather, stock, news and traffic) due to space concern. In general, both aSVM and our DNN model make good prediction on the news and stock tasks, while relatively poor on the weather and traffic tasks. Particularly, DNN-A was able to achieve 66% precision and 83% recall for the news tasks. For stock tasks, we observed that most users show consistent repeated patterns during regular stock market hours (10am to 4pm EST). Many users often repeatedly check stock quotes within some particular hour of their fa-

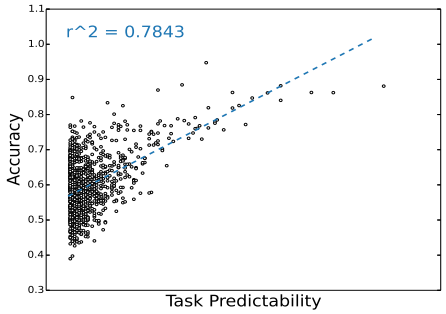


Figure 8: Task predictability vs accuracy. R^2 score indicates a strong correlation between them.

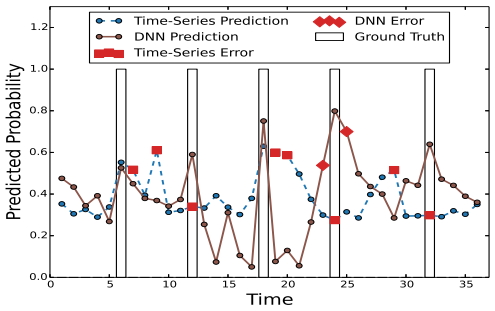


Figure 9: An Example Task and the prediction results from two models. Ground Truth = Queries issued by the user during that hour.

vorite which made the prediction easy. For news tasks, users tend to follow their day-to-day reading habit that usually occurs during morning, lunch break and evenings. Among all other categories not shown here, *games* and *sports* were also quite predictable. Note that the discovery here also confirms our previous finding in Figure 6(b) where news and stock tasks showed higher predictability scores.

Among three groups of models, we observe that target-only models in general incur much higher variance than the other two. The underlying reason is quite evident: target-only models often perform well given enough user-level data but fail miserably if there is not enough training data, which is exactly what happened to both aSVM-T and DNN-T models. In comparison, global-only models demonstrate the highest consistency across all tasks with competitive performance against target-only models on stock and traffic tasks.

Figure 9 shows an example task where our DNN approach outperforms the best time-series baseline (RNN). The task has five queries. Due to its sparsity, the time-series model made numerous error for both positive and negative classes while our DNN method only has two false positives.

5.3.4 *Predictability vs. Accuracy*

Finally, we revisit our proposed technique of calculating task predictability as discussed in Section 4.2 and 5.1. Figure 8 shows the scatter plot of task predictability and prediction accuracy for a sample of 3,000 tasks. We can easily observe that higher predictability indeed corresponds to higher accuracy, which shows the usefulness of the proposed technique. We calculate the correlation by randomly sampling the same amount of tasks for each length. This gives a 0.7843 R^2 score between predictability and accuracy, which indicates a very high correlation between these two variables.

In addition, we want to confirm our assumption that using predictability to filter out non-repeatable tasks indeed creates a cleaner training corpus. To this end, we add back those 5% tasks with low predictability scores to the training corpus and re-train a global DNN model using the Web data. We notice that both the precision and recall dropped by more than 7%, showing the crucialness of data filtering.

5.4 Ranking Performance

With the promising predictive performance of our models, we seek to apply the predictors to improve ranking in this section. Specifically, we integrate the proposed novel features and models into the production proactive system [18] to improve its ranking. Since our DNN model is optimized for classification, it is not directly applicable to compare across different types of tasks. Hence, we compare three ranking models: (1) the baseline production system using a machine learning model (Production), (2) combine the features from production and the scores of DNN model to retrain a ranker (Production+Model), and (3) combine the features from production and the model and the raw features of DNN model (as shown in Table 2) to retrain a ranker (Production+Model+Feature).

Table 8 summarizes the overall performance. Besides NDCG, we also employed a metric named First Success Position (FSP), where the success is defined as either a successful click ($>$ 30 seconds on landing pages), or a successful *view* on a task (dwell time above certain threshold on a proactive recommendation). Note that, according to previous research, the viewport-based dwell time is a good indicator of relevance for mobile devices [8] and correlates well with user eye movements [11]. For brevity, we only include results for a threshold of 10 seconds as the other thresholds (e.g., 15 seconds, 30 seconds) we tried did not make significant differences and we leave the determination of an optimal threshold as future work. In general, combining the prediction scores for each tasks with production shows noticeable performance improvement, indicating the usefulness of the proposed models. On the other hand, using only the scores of the DNN classifier for ranking turns out to be a complete failure, which supports our initial assumption that the scores are not directly applicable for ranking purpose. In addition, combining the proposed features with production also greatly improves the performance, showing that the features are capable of capturing user behavioral characteristics. Finally, by combining both the proposed model and features together with the production features, the best performance was achieved with a 14.32% improvement of NDCG@1, which is statistically significantly better than the second best-performing model with $p - val < 0.05$.

Next, we break down the ranking performance by service types for our best-performing model. Figure 10 shows the relative improvement in terms of FSP, sorted by the margin of improvement. We see that the majority of improvements lie in weather, traffic and news, while sports, local and travel services often got lower ranking in the new ranker. The reason for overall performance improvement comes from the bias towards different services. In general, users pay much more attention to news, weather and finance than other services. Therefore, improving the ranking position of these services results in better overall user experiences.

For the *weather* service, we further analyze its improvement in Figure 11, where the average success position delta

Table 7: Web to App adaptation performance breakdown for the top-4 types of tasks with standard deviation. *-G = Global Only models. *-T = Target Only models. *-A = Adaptation models.

	Weather		Stock		News		Traffic	
Methods	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
aSVM-G	0.42 ± 0.16	0.45 ± 0.17	0.46 ± 0.18	0.48 ± 0.13	0.53 ± 0.14	0.54 ± 0.11	0.40 ± 0.12	0.449 ± 0.12
DNN-G	0.44 ± 0.13	0.48 ± 0.13	0.48 ± 0.19	0.49 ± 0.18	0.53 ± 0.15	0.55 ± 0.13	0.40 ± 0.11	0.45 ± 0.13
aSVM-T	0.38 ± 0.29	0.43 ± 0.11	0.47 ± 0.22	0.48 ± 0.33	0.56 ± 0.10	0.59 ± 0.26	0.40 ± 0.17	0.38 ± 0.27
DNN-T	0.40 ± 0.19	0.44 ± 0.14	0.47 ± 0.28	0.48 ± 0.31	0.55 ± 0.17	0.58 ± 0.20	0.42 ± 0.18	0.40 ± 0.21
aSVM-A	0.47 ± 0.18	0.46 ± 0.14	0.53 ± 0.15	0.59 ± 0.11	0.63 ± 0.14	0.77 ± 0.12	0.48 ± 0.15	0.60 ± 0.12
DNN-A	**0.50 ± 0.11**	**0.51 ± 0.15**	**0.56 ± 0.14**	**0.62 ± 0.13**	**0.66 ± 0.21**	**0.83 ± 0.12**	**0.51 ± 0.18**	**0.63 ± 0.18**

Table 8: Comparison of App ranking performance over several models. FSP = First Success Position. For NDCG, higher numbers indicate better performance. For FSP, lower numbers are better. * = statistical significance with $p - val < 0.05$.

	ΔNDCG	ΔFSP
Production [18]	/	/
Model-Only	-26.28%	41.38%
Production+Model	3.07%	-5.82%
Production+Feature	9.31%	-10.95%
Production+Model+Feature	**14.32%***	**-18.97%***

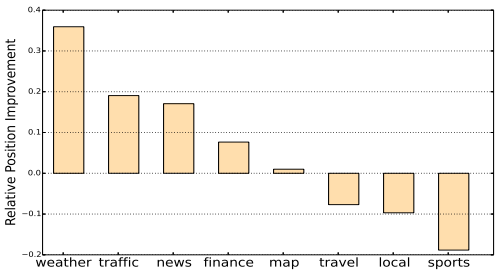


Figure 10: Relative position improvement breakdown by service types.

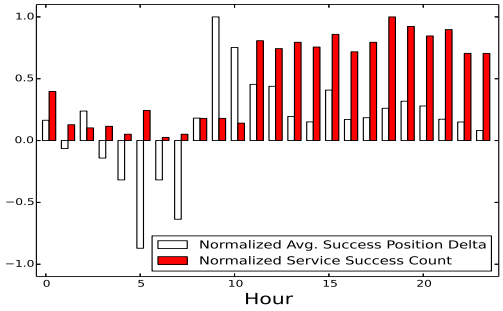


Figure 11: Weather service improvement by hour.

Table 9: Feature importance in the ranking experiment. TC = Total Clicks. TQ = Total Query Count.

Feature Name	Type	Importance	Rank
Same-hour TC past 1-D	Query-Level	1	1
TQ past 7 days	User-Level	0.065	12
TQ past 3 days	User-Level	0.033	21
Same-hour TC past 90-D	Query-Level	0.020	36
TQ past today	User-Level	0.017	43
Same-hour TC past 30-D	Query-Level	0.016	44
Time of the day	Query-Level	0.014	51

is shown for each hour. The Service count is also plotted to show the volume. It can be clearly seen that we promote the service to have gains for hours with higher success volumes (between 12pm and 23pm) and demote the service when its success is rare (between 4am and 8am).

Table 9 shows the most important features in the ranker, where *Rank* indicates the relative position of our proposed features among all production features. We can see that the new ranker picks one of our query-level features to be the most important feature, which measures the same-hour total number of clicks during the last day. It can be clearly seen that many of our features are indeed quite effective and thus treated important by the new ranker. Overall, click count from tasks and query count from users are the most useful features that improves the ranking relevance.

6. CONCLUSIONS AND FUTURE WORK

We have presented a study on understanding, characterizing and predicting user repeated search tasks in this work. We discovered from search logs and mobile App logs that certain types of user tasks such as stock-price checking often exhibited strong repetition patterns. We proposed a set of novel feature and models to predict when and what types of tasks will users issue in the near future. Despite that many previous research has been devoted to predict query or task repetition, we were among the first to take the time dimension into consideration and perform time-series prediction. In addition, we were also among the first to introduce personalized predictive models, while previous research often aggregated data across users due to insufficient user-level data. We were able to address the data sparcity issue by performing model adaptation by learning individual deep learning models for each user task. We proposed a novel metric to measure the predictability of tasks which turned out to be very effective for noise filtering of non-repeatable tasks. Experiments on a large-scale data set indicated significant performance improvement of our model when compared with other state-of-the-art time-series prediction methods. We applied the model and features to improve the recommendation of the mobile App, which showed substantial ranking improvement over the production system.

We believe that our current effort is just a small step towards a promising future of *queryless* search and recommendation engines. With more and more user search and interaction data becoming available, we will be able to make machines more intelligent so that less and less user effort needs to be spent for information discovery in the foreseeable future. Our future research directions include making prediction for a larger categories of tasks with longer timespan, leveraging similar users' behavior for new user (cold-start) prediction, as well as designing more sophisticated machine learning models for better predictive performance.

7. REFERENCES

[1] E. Adar, J. Teevan, and S. T. Dumais. Large scale analysis of web revisitation patterns. In *CHI '08*, pages 1197–1206.

[2] E. Adar, J. Teevan, and S. T. Dumais. Resonance on the web: Web dynamics and revisitation patterns. In *CHI '09*, pages 1381–1390, New York, NY, USA, 2009. ACM.

[3] E. Agichtein, R. W. White, S. T. Dumais, and P. N. Bennet. Search, interrupted: Understanding and predicting search task continuation. In *SIGIR '12*, pages 315–324.

[4] A. Aula, N. Jhaveri, and M. Käki. Information search and re-access strategies of experienced web users. In *WWW '05*, pages 583–592, New York, NY, USA, 2005. ACM.

[5] M.-W. Chang and W. tau Yih. Dual coordinate descent algorithms for efficient large margin structured prediction. *TACL*, 1:207–218, May 2013.

[6] C. L. Giles, S. Lawrence, and A. C. Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Mach. Learn.*, 44(1-2):161–183, July 2001.

[7] R. Guha, V. Gupta, V. Raghunathan, and R. Srikant. User modeling for a personal assistant. In *WSDM '15*, pages 275–284, 2015.

[8] Q. Guo, H. Jin, D. Lagun, S. Yuan, and E. Agichtein. Mining touch interaction data on mobile devices to predict web search result relevance. In *SIGIR '13*, pages 153–162.

[9] A. Hassan Awadallah, R. W. White, P. Pantel, S. T. Dumais, and Y.-M. Wang. Supporting complex search tasks. In *CIKM '14*, pages 829–838, 2014.

[10] E. Herder. Characterizations of user web revisit behavior. In *ABIS 2005*, 2005.

[11] D. Lagun, C.-H. Hsieh, D. Webster, and V. Navalpakkam. Towards better measurement of attention and satisfaction in mobile search. In *SIGIR '14*, pages 113–122, 2014.

[12] J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. *J. Mach. Learn. Res.*, 10:777–801, 2009.

[13] S. Lawrence. Context in web search. *IEEE Data Engineering Bulletin*, pages 3–25, 2000.

[14] Z. Liao, Y. Song, Y. Huang, L. wei He, and Q. He. Task trail: An effective segmentation of user search behavior. *TKDE*, 2014.

[15] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Discovering tasks from search engine query logs. *ACM Trans. Inf. Syst.*, 31(3), Aug. 2013.

[16] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, and G. Tolomei. Modeling and predicting the task-by-task behavior of search engine users. In *OAIR '13*, pages 77–84, 2013.

[17] C. Sheng, J. Zhao, Y. Liu, and W. Wang. Prediction for noisy nonlinear time series by echo state network based on dual estimation. *Neurocomput.*, 82:186–195, Apr. 2012.

[18] M. Shokouhi and Q. Guo. From queries to cards: Re-ranking proactive card recommendations based on reactive search history. In *SIGIR '15*, 2015.

[19] Y. Song, H. Ma, H. Wang, and K. Wang. Exploring and exploiting user search behavior on mobile and tablet devices to improve search relevance. In *WWW '13*, pages 1201–1212, 2013.

[20] Y. Song, H. Wang, and X. He. Adapting deep ranknet for personalized search. In *WSDM '14*, pages 83–92, 2014.

[21] H. Wang, X. He, M.-W. Chang, Y. Song, R. W. White, and W. Chu. Personalized ranking model adaptation for web search. In *SIGIR '13*, pages 323–332, 2013.

[22] H. Wang, Y. Song, M.-W. Chang, X. He, R. W. White, and W. Chu. Learning to extract cross-session search tasks. In *WWW '13*, pages 1353–1364, 2013.

[23] Y. Wang, X. Huang, and R. W. White. Characterizing and supporting cross-device search tasks. In *WSDM '13*, pages 707–716, New York, NY, USA, 2013. ACM.

[24] R. W. White, P. Bailey, and L. Chen. Predicting user interests from contextual information. In *SIGIR '09*, pages 363–370, New York, NY, USA, 2009. ACM.

[25] R. W. White, W. Chu, A. Hassan, X. He, Y. Song, and H. Wang. Enhancing personalized search by mining and modeling task behavior. In *WWW '13*, pages 1411–1420.

[26] J. Yan, W. Chu, and R. W. White. Cohort modeling for enhanced personalized search. In *SIGIR '14*, pages 505–514, New York, NY, USA, 2014. ACM.

[27] J. Yang, R. Yan, and A. G. Hauptmann. Cross-domain video concept detection using adaptive svms. In *MULTIMEDIA '07*, pages 188–197, 2007.

[28] M. Yang. *Statistical models for count time series with excess zeros*. PhD thesis, University of Iowa, 2012.

[29] R. Zhang, Y. Konda, A. Dong, P. Kolari, Y. Chang, and Z. Zheng. Learning recurrent event queries for web search. In *EMNLP '10*, pages 1129–1139, 2010.