Efficient Behavior Prediction Based on User Events

Péter Szabó, and Béla Genge

***Abstract*—In 2020 we have witnessed the dawn of machine learning enabled user experience. Now we can predict how users will use an application. Research progressed beyond recommendations, and we are ready to predict user events. Whenever a human interacts with a system, user events are dispatched. They can be as simple as a mouse click on a menu item or more complex, such as buying a product from an eCommerce site. Collaborative filtering (CF) has proven to be an excellent approach to predict events. Because each user can generate many events, this inevitably leads to a vast number of events in a dataset. Unfortunately, the operation time of CF increases exponentially with the increase of data-points. This paper presents a generalized approach to reduce the dataset's size without compromising prediction accuracy. Our solution transformed a dataset containing over 20 million user events (20,692,840 rows) into a sparse matrix in about 7 minutes (434.08 s). We have used this matrix to train a neural network to accurately predict user events.**

***Index Terms*—behavior prediction, machine learning, collaborative filtering, user events, Adam.**

I. Introduction

One of the goals of user experience (UX) research is understanding and predicting how humans (the users) will interact with a computer system [1], [2]. When a user interacts with any digital system, a series of events is being dispatched and stored (logged). For example, an event could be buying or viewing an item on an e-commerce website, starting to watch a movie on a streaming platform, or even something as simple as clicking on a website's menu item.

To predict future events a user might dispatch, a new research topic emerged, using unsupervised machine learning (ML) to create better UX [3]. Collaborative filtering (CF) has proven to be an excellent approach to predict events [4]–[6]. CF algorithms calculate the probability of a user's future events based on the events dispatched by other users. Traditionally, CF algorithms were used to predict the rating a user would give to an item, for example, a movie, based on the star rating from other users [7]. Unfortunately, most events do not have ratings associated with them. Even where it would be possible, such as buying an item from an online store, users seldom leave ratings. Moreover, there are many scenarios where we cannot rationally expect ratings.

The foremost problem researchers faced when trying to predict events was that it inevitably leads to a vast number of events. Because the operation time of recommender engines increases exponentially with the increase of data-points [8] we need a way to reduce the training dataset in a way which minimizes the effect of data reduction on the prediction accuracy.

At SoftCOM 2020, we presented a solution reducing all events between an item in an eCommerce shop and a user (the event chain) to a single UX value in the $[0, 1)$ range. This number also depends on the sequentiality of the events. After determining the UX value for all event chains, we store it in a sparse matrix. This matrix can be generated from any event dataset in linear time. Using a random split of the sparse matrix, we trained a deep neural network to predict unknown (null) values. We have implemented a machine learning model featuring Adam optimizer, and dropout regularization [9].

In this paper, we present the generalization of the previous research. Instead of items, this generalized method will work on any event type and any object regardless of context. An event's object can be an item from an eCommerce site, a function of software, or even a game-play choice in a computer game. Hereinafter, we call them objects, although they can be conceptually and programmatically different entities.

We assess the generalization experimentally with a dataset consisting of 123 unique event types, with data from most of 2020 (January 5 to November 21) provided by the REES46 Marketing Platform on Kaggle [10].

Moreover, in the experimental assessment, we prove the method's scalability, with another dataset from REES46 [11]. This dataset contains over 20 million user events (20,692,840). Our method transforms this 'gargantuan' dataset into a sparse matrix ready for machine learning in about 7 minutes (434.08 s) on our test machine.

We have structured the remainder of this paper as follows: After an overview of related works in Section II, we presented the developed approach in Section III, followed by Section IV, which documents the numerical evaluation of the developed approach, applied to multiple real-world datasets. We concluded the paper with Section V.

II. Related Works

Relying on user events to create better experiences, usually in the form of recommendations, has become a popular research topic, and its popularity continues to increase in 2020. Some of the papers which inspired the research presented in this paper include Moscato, *et al.* creating a recommender system based on the personality traits, moods, and emotions of a user [12]. Next, Szabo *et al.* used ML, and behavioral

Manuscript received December 3, 2020; revised May 14, 2021. Date of publication May 25, 2021. Date of current version May 25, 2021.

Authors are with the George Emil Palade University of Medicine Pharmacy Science and Technology of Targu Mures, Romania (e-mails: peter@kaizen-ux.com, {peter.szabo, bela.genge}@umfst.ro).

Part of this work was presented at the International Conference on Software, Telecommunications and Computer Networks, SoftCOM '20, Hvar, Croatia, September 17-19, 2020.

Digital Object Identifier (DOI): 10.24138/jcomss-2020-0011

analysis for user-tailored viewer experience for online video streaming [13]. Hug published 'Surprise', a Python library for recommender systems [14]. Beheshti *et al.* presented a cognitive recommender system, which does not rely on domain experts' knowledge to adapt to new situations [15].

Real-world recommender engines often rely on data reduction to solve scalability and enhanced generalization by reducing over-fitting. This step is essential for all 'gargantuan' datasets but can easily lead to low accuracy [16]–[18]. This paper proposes a solution which efficiently reduces the data to speed up training without sacrificing much of the accuracy, as proven experimentally in Section IV

Neural CF needs hyper-parameters to control the learning process, similarly to other neural network based machine learning. Hyper-parameter optimization (HPO) aims to select the hyper-parameters so that the loss function converges to the minimum much faster, essentially making the learning faster or more accurate given a time constraint [19], [20]. CF research papers usually omit HPO, probably because traditional HPO can take a considerable amount of time. In 2019 Dacreama, *et al.* [21] while evaluating commonly used CF datasets, found that they contain only a few hundred thousand ratings. For those 'tiny datasets', as Dacrema *et. al.* calls them, HPO 'can take days or weeks.' Therefore 'gargantuan' datasets, such as the ones from Section IV of this paper, HPO could take a considerable amount of time. To address this, in 2020 by Szabo and Genge presented a hybrid hyper-parameter optimization (HPO) solution to find the best hyperparameters for CF featuring an Adam optimizer-based or one of the other modern stochastic gradient descent (SGD) optimizers. This research also has implications for our current research because it proves the problem's practical solvability in a scalable manner for gargantuan datasets [22]. That paper shows that the learning algorithm converges to the minimum in very few batches, especially with hybrid HPO. The researchers got usable results after the first 30 minutes because of the fast convergence even after the data reduction.

One of the shortcomings of the traditional CF system is called the 'cold-start problem,' meaning that new items do not get predictions meaningful until a few users interact with them [23]. For rating-based recommenders, the cold-start problem can mean weeks of delay. We find several suggestions for alleviating these issues [24]–[26]. Castillejo *et al.* suggests using data from the users' social network in a CF recommender system [27] to tackle the cold start problem. Dong *et al.* described a solution for cold start named 'MAMO' (Memory-Augmented Meta-Optimization) with two memory matrices. The feature-specific matrix for personalized parameter initialization, while the task-specific matrix gives a fast prediction of the user preference [28].

Data sparsity is another problem with traditional CF, meaning that we lack data for a very high percentage of user-item pairs. Natarajan *et al.* suggested resolving data sparsity and cold start problem using Linked Open Data [29]. Zhang *et al.* suggested clustering nodes in bipartite networks to tackle data sparsity [30].

Contrary to the aforementioned papers, our approach addresses the cold-start and data sparsity problem by relying on user events. New users start dispatching events as soon as they start using the application and new objects get events with no noticeable delay, so cold-start has almost zero impact on our method. Moreover, event data is created and stored in abundance. Virtually all users and objects will have significant events within minutes, so data sparsity is not an issue.

The methodology documented in this paper distinguishes itself by presenting a scalable solution to create accurate predictions of user events. One of the significant contributions is the generalization of the UX value function, a data reduction solution, which enables relatively fast learning speeds and accurate event predictions, as demonstrated in Section IV.

III. Developed Approach

Our solution predicts the probability of an event happening for a user concerning an object. For example, the likelihood of the user 'Jane' buying a 'smartphone' in a shop, or 'Joe' clicking on the 'red car' in a game.

We summarise the developed approach as the subsequent execution of the following steps:

1) Calculate the event-type likelihood matrix
2) Choose an event-type to predict
3) Compute the User Experience (UX) value for all user-object event chains for the event-type we want to predict
4) Store the UX values as a user-object sparse matrix
5) Train a neural network to predict the likelihood of all events for unknown user-object pairs with minimal mean squared error

In order to formally define the approach, we first introduce the following notations: Let E denote the ordered set of all user events (the dataset), and $e \in E$, an element of E. Let e_i denote the i^{th} user event in E. It is assumed that given two user events e_i, and e_j, if $i < j$, then user event e_i precedes in time user event e_j.

We categorize the events into meaningful categories based on their similarities. We call those categories an *event types*, and denote the k^{th} event type as T_k. Each event in E will be associated with one and only one event type, so that $E_i \in T_k$. Multiple events can be associated with the same event type. In fact, we want to avoid creating event types with a low number of events in them, as they are less useful in practice. Let $|T|$ be the cardinality of T. In other words, we will have $|T|$ number of event types.

A. Calculate the Event-type Likelihood Matrix

Let $\mathrm{P}(T_k)$ be the probability of the event type T_k in the complete sample space, and $\mathrm{P}(T_k|T_l)$ the probability of event type T_k given that the event type T_l has already occurred. We assume that the probability of event type T_k is dependent of the T_l event type. The event-likelihood matrix can be defined as:

$$\forall T_k (T_k \in T \implies \mathrm{P}(T_k) \neq \mathrm{P}(T_k|T_l))$$
$$P(T_k|T_l) = \begin{cases} 0, & \textit{if}\ \mathrm{P}(T_l) = 0 \\ \frac{\mathrm{P}(T_k \cap T_l)}{\mathrm{P}(T_l)}, & \textit{if}\ \mathrm{P}(T_l) \in (0,1]. \end{cases} \quad (1)$$

It is worth mentioning that the main diagonal should always be 1, and ideally, all other probabilities should be less than 1.

If any event type T_k occurs with a probability of 1 or near that number, given another event T_l, we eliminate event type T_k, as redundant, as summarised in the following equations:

$$\begin{aligned} k = l &\implies \mathrm{P}(T_k|T_l) = 1 \\ k \neq l &\implies \mathrm{P}(T_k|T_l) \ll 1 \end{aligned} \tag{2}$$

B. Choose an Event-type to Predict

We need to choose an event-type we want to predict. For example, buying an item or subscribing to a newsletter. Let T_c be the event type we choose to be predicted so that $T_c \in T$. The event-type likelihood matrix will contain the probability of the chosen event, as defined in (1).

C. Compute the UX Value for an Event Chain

All events between the same user and the same object form the UX event chain, where events are ordered based on their recency, starting with the oldest event. The UX value function transforms the most recent element of the UX event chain into a scalar value.

Let u denote a specific user, and o a specific object. Let $E^{u,o} \subseteq E$ denote an event chain associated to user u and specific to object o. Then, let $E^{u,o}_i \in E^{u,o}$ be the i^{th} element of $E^{u,o}$. The members of the $E^{u,o}$ event chain differ only in event type. Let the event type of the given event be $T_k \in T$, the k^{th} element of T. As a result, the UX function is defined as:

$$UX(E^{u,o}_i) = \begin{cases} 0, & \text{if } i = 0. \\ \tanh(UX(E^{u,o}_{i-1}) + \mathrm{P}(T_c, T_k)), & \text{if } i > 0. \end{cases} \tag{3}$$

Similarly to the specific method described by us previously [9], the hyperbolic tangent is used to assure that the function's return value is always in the $[0, 1)$ range, acting as the UX value's normalizer.

D. Store the UX values

We define the $Y_{u,o}$ sparse matrix to contain the UX value for the last event in the $E^{u,o}$ event chain, as defined in the following equation:

$$Y_{u,o} \Leftarrow UX(E^{u,o}_{max(i)}) \tag{4}$$

Our approach's main benefit is that it has a computational complexity of $O(|E|)$, where $|E|$ is the number of events. In other words, it has a linear computation time. Another benefit of the approach is that it yields a sparse $user \times objects$ matrix, most of the values being zero, which can be constructed and stored efficiently.

Let ϖ be the point above which a value is likely to result in an event happening. As we will see in the experimental assessment, this value is not necessarily 0.5. Moreover, ϖ is not used for training or data preparation, but it will help us understand the predictions.

We introduce the term 'worst rational case number of events,' meaning the largest number of events we can rationally expect to happen between the same user and the same object in a row. For example, we can expect at most one remove from cart event. It doesn't make sense to remove the object from the cart right after it was removed. At least one add to cart event has to happen before the removal can happen again. Similarly, a play event for a movie can't happen right after the play event for the same film. For other events, it is possible to have multiples in a row. For example, a user can click add to cart after adding to cart. In theory, this can happen an infinite number of times, but from the observation of real-world datasets, such as [11] we can conclude that users will rarely do this, and at most, three times.

Let $W \in \mathbb{N}$ denote the worst rational case number of events of a certain event type T_k, affecting ϖ. W can be calculated with the empirical formula $W = \lfloor \mu + 3 \times \sigma \rfloor$. We calculate ϖ using the following equation:

$$\begin{aligned} &\varpi = UX(T_c) + W_1 UX(T_1) + \dots W_n UX(T_n) \\ &n \leq |T|\,;\; n \neq c \end{aligned} \tag{5}$$

E. Train a Neural Network

To train a neural network, we split the ground truth Y sparse matrix into three matrices with the same dimensions as Y using a three-way data split.

For colossal datasets, holdout validation is recommended by many authors [31], [32]. We used a three-way data split, a common variant of holdout validation. This means that only the trained model is evaluated against the test set, and the validation set is used during HPO only.

Let p_{train}, p_{val}, and p_{test} be the probability of an event chain being in the Y_{train}, Y_{val}, and Y_{test} matrices respectively, we apply three-way data split using the following equation:

$$\begin{aligned} &p_{train} + p_{val} + p_{test} = 1 \\ &p_{train} Y_{train} + p_{val} Y_{val} + p_{test} Y_{test} = Y \end{aligned} \tag{6}$$

We need to train a neural network (NN) to predict the likelihood of conversion for all the user-object pairs with unknown UX values, represented as zero (which is ideal for a sparse matrix).

Vavasis [33] proved that non-negative matrix factorization is NP-hard. Therefore, it is unlikely that there is an exact algorithm that runs in polynomial time. Algorithms that run in time exponentially cannot reasonably be considered for real-world applications due to the datasets' large dimensions. The deep neural network described below can approximate the values with outstanding accuracy as demonstrated in the experimental assessment, in section IV.

To predict the zeroes' non-zero values in the sparse matrix, we define the objective function, a loss function to be minimized. Due to the UX value function described above, there can be no outliers, as all values will fall in a relatively narrow range, depending on the initial set of events. On the other hand, the resulting data will have a small perturbation. This small perturbation will significantly affect mean absolute error (MAE), compared to mean squared error (MSE). Therefore we have chosen MSE as our loss function.

Let $\hat{Y}$ store be q predictions about Y ground truth, as calculated by our solution. We apply the MSE function to the q data points. Hereinafter we will call this MSPE, short for mean squared prediction error, defined in the following equation:

$$MSPE(\hat{Y}, Y) = \frac{1}{q} \sum_{a=n+1}^{n+q} (Y_a - \hat{Y}_a)^2 \tag{7}$$

Let $backpropagation()$ be a function for backward propagation of errors [34] and let $Adam(lr, \beta_1, \beta_2)$ be the Adam optimizer function to minimize MSPE. The Adam optimizer, as introduced by Kingma *et al.* [35]. The Adam optimizer combines the benefits of AdaGrad and RMSProp algorithms, and it can handle sparse gradients on noisy problems. It was proven to be a suitable optimizer for many modern problems, including CF [36]–[38]. To use it, we need to define its hyper parameters as such: Let $lr \in (0, 1)$ be the learning rate, and $\beta_1, \beta_2 \in [0, 1)$ be the decay rates for moment estimates.

Let $epochs \in \mathbb{N}_{>0}$ denote the number of times the entire dataset Y_{train} is passed both forward and backward through the NN. Let $batch\_size \in \mathbb{N}_{>0}$ denote the number of samples evaluated before the model's internal parameters are updated. Our algorithm will run until Y_{train} is passed forward and backward through the NN $epochs$ times. During each pass, $\frac{Y_{train}}{batch\_size}$ number of batches are taken from Y_{train}.

It's worth noting that L_2 regularization is not effective to prevent overfitting in the case of Adam, as demonstrated by Loshchilov *et al.* [39]. We could use decoupled weight decay regularization [39] or dropout [40]. Weight decay penalizes large weights, forcing all weights to be close to 0. Because we work with gargantuan datasets, dropout regularization is a much better option, because it means dropping units and their connections from the neural network during training. Let $D()$ be a dropout regularization function, and $P_0 \in [0, 1)$ the probability of eliminating units and their connections from the NN during training [40], [41].

Let $Embedding(x, y)$ be a lookup function that retrieves embeddings. Let x be the size of the dictionary of embeddings, and y be the size of each embedding vector. Let $n_f \in \mathbb{N}_{>0}$ be the number of factors. Let $user_f$ be the user factors, and $users$ are the users found in the batch, so that $user_f \Leftarrow Embedding(users, n_f)$. Let $user_b$ be the user bias, so that $user_b \Leftarrow Embedding(users, 1)$. Moreover let obj_f be the object factors, and obj the objects found in the batch, so that $obj_f \Leftarrow Embedding(obj, n_f)$. Let obj_b be the object bias, so that $obj_b \Leftarrow Embedding(obj, 1)$.

In the developed algorithm the loss is calculated using the MSPE function (Eq. (7)). Then, the $backpropagation()$ achieves the backward propagation of errors. Finally, the Adam optimizer function is used to minimize the loss function.

After each epoch, the validation loss is calculated and stored using $MSPE(\hat{Y}, Y_{val})$. When we observe a decreasing validation loss, we call this 'learning', because our solution is getting closer and closer to predicting the events. The resulting solution is summarized as Algorithm 1.

Algorithm 1 CF object prediction algorithm

```
function CF($Y_{train}, Y_{val}, lr, \beta_1, \beta_2, batch\_size, n_f, P_0$ )
    for $i \leftarrow 1, epochs$ do
        for all $batch \in \frac{Y_{train}}{batch\_size}$ do
            $user_f \Leftarrow Embedding(users, n_f)$
            $object_f \Leftarrow Embedding(obj, n_f)$
            $user_b \Leftarrow Embedding(users, 1)$
            $object_b \Leftarrow Embedding(obj, 1)$
            $\hat{Y} \Leftarrow \sum(D(user_f) \times D(obj_f)) + user_b + obj_b$
            $loss \Leftarrow MSPE(\hat{Y}, Y_{train}))$
            $backpropagation(loss)$
            $Adam(lr, \beta_1, \beta_2)$
        $loss_{val} \Leftarrow MSPE(\hat{Y}, Y_{val}))$
        return $loss_{val}$
```

IV. Experimental Assessment

A. Implementation Details

We have used Python (version 3.8.5), SciPy (version 1.5.2), PyTorch (version 1.7) with CUDA 11.0, and the PyTorch Lightning lightweight wrapper [42] (version 1.0.8) to implement the solution described in the previous section. The Jupyter Notebooks can be found on the project's repository on GitHub: https://github.com/WSzP/uxml-ecommerce

The test machine we used had the following configuration: Intel Core i9-9900K CPU (3.60GHz); 64 GiB RAM and NVIDIA GeForce RTX 2080 Ti GPU for GPU based ML.

B. The Datasets

To test the scalability and generalizability of the developed approach, we have used two 'gargantuan' datasets with real-world user event data.

The first dataset is the 'eCommerce Events History in Cosmetics Shop' dataset [11], an extended version of the dataset used by us [9]. This extended version contains user data from five months of user data instead of the two used previously, resulting in 20,692,840 events instead of 8,738,120. The shape of the dataframe is 20692840×9. The data preparation procedure used for dataset 1 is summarised as Fig. 1.

The second dataset we used in our experimental assessment is the 'eCommerce purchase history from electronics store' dataset [10]. This is a dataset unlike any other on Kaggle because it covers purchases from a multi-category eCommerce site, which happened in 2020 (January 5 to November 21 at the time of writing this paper). This dataset was not solvable with the method described in our previous paper [9], but as we will demonstrate, it is possible to predict purchases of a category of items, based on the purchase behavior demonstrated in other categories, so the event types will be the item categories for which we have enough purchase data. The data preparation procedure used for dataset 2 is summarised as Fig. 2.

C. Data Cleansing and Reduction

When working with raw data coming directly from user logs, first, we need to do a data reduction and cleansing. This

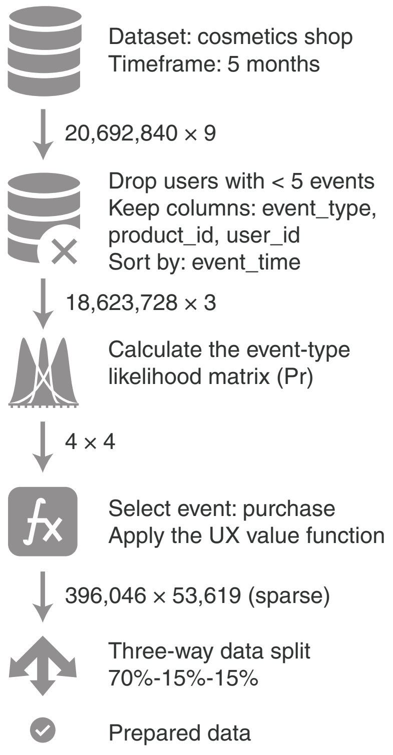
Dataset: cosmetics shop
Timeframe: 5 months
20,692,840 × 9
Drop users with < 5 events
Keep columns: event_type, product_id, user_id
Sort by: event_time
18,623,728 × 3
Calculate the event-type likelihood matrix (Pr)
4 × 4
Select event: purchase
Apply the UX value function
396,046 × 53,619 (sparse)
Three-way data split
70%-15%-15%
Prepared data

Fig. 1. Data preparation procedure used for dataset 1

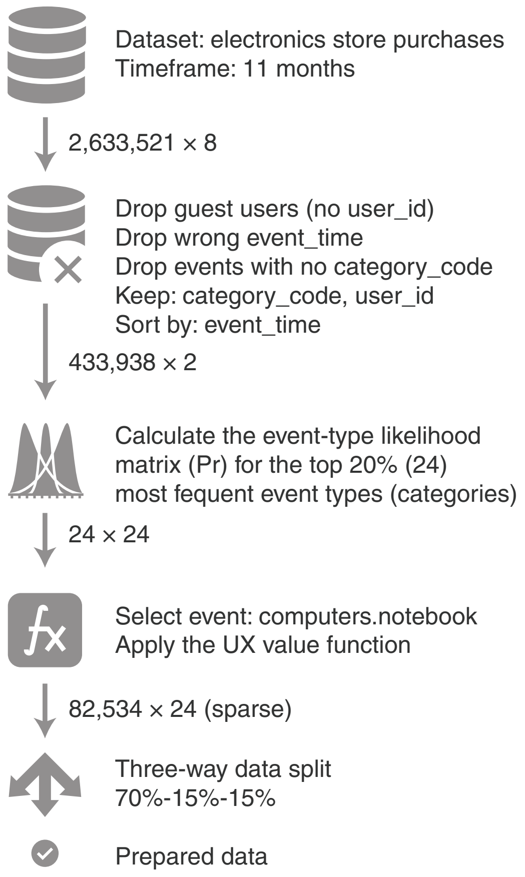
Dataset: electronics store purchases
Timeframe: 11 months
2,633,521 × 8
Drop guest users (no user_id)
Drop wrong event_time
Drop events with no category_code
Keep: category_code, user_id
Sort by: event_time
433,938 × 2
Calculate the event-type likelihood matrix (Pr) for the top 20% (24) most fequent event types (categories)
24 × 24
Select event: computers.notebook
Apply the UX value function
82,534 × 24 (sparse)
Three-way data split
70%-15%-15%
Prepared data

Fig. 2. Data preparation procedure used for dataset 2

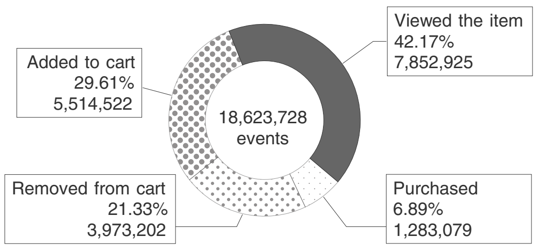

Fig. 3. User events in dataset 1 (percentage and number).

requires a different approach for different datasets; therefore, we describe it separately.

We drop all events associated with users for the first dataset, who had less than five events in the dataset. This means that they have interacted with less than five objects on the site during the whole period of five months. Five events can happen during the first minute of their interaction with the software, so it is pointless to predict their next events. Even if we have created some predictions for them, it is unlikely that they would come back to the site to benefit from the enhanced user experience. When we dropped 1,243,312 (75.84%) of users for not meeting the minimum 5 event requirement, we have reduced the number of total events by 10%. This does not seem like a huge gain, but it also leads to a 75.84% reduction of the sparse matrix's user dimension, which leads to a significant speed gain. The distribution of event types is plotted as Fig. 3.

For the second dataset, we can easily identify guest users. We need to remove them because we cannot predict the behavior of unknown users. (In some cases, we can deduce the identity of the guest users, but this is beyond the scope of this paper, and we assume that by not logging in, the user intent is to remain anonymous and not have a personalized experience.) After removing them, we reduce our data frame's size from 2,633,521 rows to a much more manageable 564,169 rows. Then we remove events with the obviously erroneous timestamp (as all data is from 2020, any timestamp with a different year is an error). This leaves us with 562,862 rows. Then we drop events with missing category references, leading to 433,938 rows. In dataset 2, the event types are the categories. We will find 123 unique categories in this cleaned-up dataset. Our recommendation is to reduce the number of events to a 20% if there are many event types, discarding the bottom 80% when ordered by the number of events in each category. If we omit this step, for the second dataset, we encounter situations, such as the probability of buying 'electronics.smartphone' given 'apparel.shoes' being 1. This results from only two items in the 'apparel.shoes' category being sold in 2020, and they were purchased by the same user (user_id: 1515915625512096000). Because this user bought a smartphone, the probability is correct, but it is obvious, that buying a shoe will not automatically mean that one will also buy a smartphone. Anomalies such as this result from a rare events being included in the dataset (such as buying shoes from an electronics store). The final distribution of event types is plotted as Fig. 3.

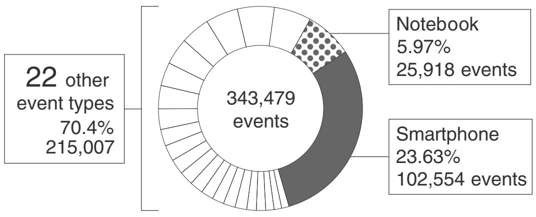

Fig. 4. User events in dataset 2 (percentage and number).

TABLE I
RUNTIME EVALUATION OF THE DATA PREPARATION (DATASET 1)

	first 2 months	all 5 months	Change
raw data (number of rows)	8,738,120	20,692,840	237%
unique users	177,592	396,046	223%
unique objects	44,780	53,619	120%
data loading	4.65s	11.2s	241%
UX value function	165.6s	434.1s	262%
three-way datasplit	60.2s	152.1s	253%

D. Calculating the Event-type Likelihood Matrix

Using equation (1), we compute the event-likelihood matrix. The runtime evaluation of the UX value function with two other functions of the same datasets is published here as Table I and II. This proves that the UX value function does not take considerably more time than other common tasks on the same data frame, such as data loading or data split.

E. Exploring the UX Value Function

The user experience value function's ultimate goal is to predict if a user even will happen or not, in other words, if it is above the activation threshold value or not. With this threshold, we can also treat the problem as a classification problem, essentially dividing event-user pairs into likely happening and likely not happening groups. The threshold itself represents the probability which we treat as the minimum requirement for us to have confidence in the even's happening for a given user.

To solve this problem more transparently, we have introduced a notation for the event chain, an array of events. In this array, capital letters denote the event. For example, 'V' for view event.

This way, we can refer to any event chain with a simple array notation. For example, if a user views a product, then adds it to the cart, and finally purchases it, the event chain is $[V, C, P]$. In this example, the UX value of the last event in the chain returns 0.8464209. This is the value calculated for the first dataset's first two months. When not specified, we are referring to this dataset in the later parts of this experimental assessment. For events encountered multiple times, an index notation is implemented to save space, denoting the consecutive occurrence of an event, for example, $[V, V, V] \Rightarrow [V^3]$.

With this notation, we have provided a few common example chains, with the approximate UX value of the last event in the chain given in Table III. Because $\lim_{n\to\infty} UX(V^n) \approx 0.51083773$, the point above which an event is positive can't be 0.5, let alone $\tanh(0.5) \approx 0.46$. No matter how many times a user views an item, the function will not predict a purchase. On the other hand, $UX(P) \approx 0.7615942$ cannot serve as the point above, which we consider the event positive. For example, if a user has the event chain $[V, C, P, V^2]$ they purchased the item, because of $P \in [V, C, P, V^2]$, but the UX value of the last element of the chain $[V, C, P, V^2]$ will be 0.6463784, which is smaller than $UX(P) \approx 0.7615942$.

To find the ϖ value for the dataset, we analyzed the events in relation to the purchase time, if they happened before or after. It was found that most events regarding a specific user and item pair happen before purchase, as summarized in table IV.

When the event chain has the form of $[e^n, P]$, where e denotes any event, and $n \in \mathbb{N}$, the UX value of the chain will be at least $UX(P)$, so there is no need to consider pre-purchase events.

We calculate the mean (μ) and standard deviation (σ) of the after-purchase event arrays for each event type. Then we construct a worst-case scenario event chain, taking the events in the increasing order of value, and applying the three-sigma rule. The three-sigma rule stipulates that nearly all values lie within three standard deviations of the mean. When analyzing human behavior, the three-sigma probability can be treated

TABLE II
COMPARISON OF DATASET 1 (FIRST 2 MONTHS) AND DATASET 2 (11 MONTHS)

	Dataset 1 (first 2 months)	Dataset 2 (all 11 months)	Change
raw data (number of rows)	8,738,120	2,633,521	30%
unique users	177,592	82,534	46%
unique objects	44,780	24	0.05%
data loading	4.65s	0.13s	2.77%
UX value function	165.6s	9.4s	5.65%
three-way datasplit	60.2s	3.0s	4.92%

TABLE III
UX EVENT CHAINS AND THE APPROXIMATE UX VALUE OF THE LAST ELEMENT IN THE CHAIN

Event chain	UX value of the last item
$[P]$	0.7615942
$[C]$	0.1929539
$[V]$	0.052975323
$[V^2]$	0.10560506
$[V^3]$	0.1573127
$[V^17]$	0.49882156
$[V^18]$	0.5019028
$[V^50]$	0.51083726
$[V, C, P]$	0.8464209
$[V, C, R, C, P]$	0.89389783
$[V, C, R, V, C, P]$	0.90029126
$[V, C, P, V^2]$	0.6463784
$[V, C, P, V, C, R]$	0.64342976
$[P, C^2, V, R]$	0.6038386

TABLE IV
EVENTS BETWEEN A SPECIFIC USER AND ITEM HAPPENING BEFORE AND AFTER PURCHASE.

Events	Before purchase	After purchase
V	3,883,216 (98.60%)	55,080 (1.40%)
C	2,509,881 (98.65%)	34,311 (1.35%)
R	1,668,179 (98.85%)	19,412 (1.15%)

TABLE V
WORST RATIONAL CASE CALCULATOR FOR EACH EVENT TYPE, AFTER PURCHASE

add to cart events after purchase Mean (μ) = 0.06040233011349533 Standard deviation (σ) = 0.7254660441273392 $W = \lfloor \mu + 3 \times \sigma \rfloor$ = 2
view events after purchase Mean (μ) = 0.09696483176390437 Standard deviation (σ) = 0.5475150110268789 $W = \lfloor \mu + 3 \times \sigma \rfloor$ = 1
remove from cart events after purchase Mean (μ) = 0.03417358958244211 Standard deviation (σ) = 0.3552707937839338 $W = \lfloor \mu + 3 \times \sigma \rfloor$ = 1

as near certainty [43]. The calculations' results, including W values are summarized in table V.

As a result of the worst-case calculations, we have reached the worst rational case event chain: $[P, C^2, V, R]$. This exact event chain is improbable, but any number above that can be treated as a positive prediction, in other words, the user is likely to purchase the given item. In contrast, any number below this can be treated as a negative prediction, meaning that the user is unlikely to purchase the item. The ϖ value for the dataset was calculated to be 0.6038386. Please note that this value needs to be recalculated for each implementation. A different project might have a different ϖ value.

F. Evaluation Metrics

For testing the machine learning models' predictions and the data-reduction itself against the ground truth, twelve evaluation metrics were used. The goal of those metrics for an event-based predictor is to evaluate how accurately the engine predicts the user's events. The metrics discussed below were implemented as https://github.com/WSzP/uxml-ecommerce/blob/master/test-uxml.ipynb.

When it comes to metrics, different sources use different metrics to measure recommender systems. For example, Hu *et al.* uses full dataset precision, recall, and F_1 [44]. In contrast, McInerney *et al.* prefers Normalized discounted cumulative gain and expected stream rate [45] (expected stream rate is not relevant for our e-commerce datasets for obvious reasons). We included all known metrics to improve the research's comparability, even those metrics, which are more-or-less irrelevant for event-based prediction, as long as they can be calculated with reasonable effort.

It is worth noting that we have used multiple methods to calculate most metrics to ensure that no calculation method results in a significantly different result. As all methods resulted in approximately the same value, there is no need to list the slightly different values resulting from different methods in this paper's following results.

As mentioned in Subsection III-E, mean squared error, being the loss function for training and validation, was the first metric considered. For completeness, root mean squared error is also given for test results in this paper, but that is simply the square root of the MSE. For testing purposes, the mean absolute error was calculated using Eq. (8) for q data points. Please note that for comparing MSE, RMSE, and MAE results, the smaller value is better.

$$MAE(\hat{Y}) = \frac{1}{q} \sum_{i=n+1}^{n+q} |Y_i - \hat{Y}| \tag{8}$$

The previously described activation threshold enables us to use metrics usually applicable to classification problems. The whole dataset's confusion matrix is created by counting the true positive, negative, and false-positive and negative values between $\hat{Y}$ and Y.

The full test dataset precision can be calculated by dividing the number of true positives by the number of all positive results, as shown in equation (9). The full test dataset recall can be calculated by dividing the number of true positives by the total of true positives and false negatives, as shown in equation (10). Finally, F_1 is the harmonic mean of precision and recall (equation (11)). Precision, recall, and F_1 values are considered better if the results are closer to 1.

$$Precision(\hat{Y}, Y) = \frac{\sum_{i=n+1}^{n+q} |\hat{Y}_i \bigcap Y_i|}{\sum_{i=n+1}^{n+q} |\hat{Y}_i|} \tag{9}$$

$$Recall(\hat{Y}, Y) = \frac{\sum_{i=n+1}^{n+q} |\hat{Y}_i \bigcap Y_i|}{\sum_{i=n+1}^{n+q} |Y_i|} \tag{10}$$

$$F_1(\hat{Y}, Y) = 2\frac{Precision(\hat{Y}, Y) \times Recall(\hat{Y}, Y)}{Precision(\hat{Y}, Y) + Recall(\hat{Y}, Y)} \tag{11}$$

G. Testing the Data-reduction Against the Ground-truth

We analyzed the results of the data preparation against the ground truth to evaluate how accurately the UX event chain function represented the multitude of user events with a single value. In other words, how much information is lost due to reduction.

For the close to zero MAE (0.0036) and RMSE (0.0602) proves that the result of data reduction can be used as a basis of training. Our assumption is reinforced by having only 0.3% false positives. Precision is 0.9827, recall 0.9930, while $F_1 = 0.987$. The radius of the 95% confidence interval for the squared prediction error is 0.00063.

The results of the tests are detailed in Table VI. Close to zero mean absolute error and root mean squared error proves that the result of data reduction can be used as a basis of training. Our assumption is reinforced by having only 0.3% false positives. The false negatives are also low (0.7%), but they are of little concern for this research because a false negative predictions would not be shown in recommendation.

TABLE VI
DATA REDUCTION RESULTS TESTED AGAINST GROUND TRUTH.

Metric	Result
RMSE	0.060224
CI 95%	0.0595, 0.0608
MAE	0.003626
true positive	99.30%
false negative	0.70%
false positive	0.30%
true negative	99.70%
precision	0.982771
recall	0.993041
F_1	0.987879

V. CONCLUSIONS

The solution presented in this paper predicts user events based on previously recorded user events from all users. One of this paper's main contributions is the generalized UX value function, a method to reduce all events between a user and any object to a single scalar value in the $[0, 1)$ range while accounting for the sequentiality of the events. Calculating this UX value for any event type and dataset runs in linear time. The machine learning model illustrates this method's usefulness by demonstrating that ML will achieve a fast learning speed and good prediction accuracy when learning from the UX value sparse matrix.

As future research, we intend to explore further data reduction possibilities for data-sets containing user events. Moreover, we hope to improve the machine learning algorithm used to achieve consistent convergence in fewer iterations. Moreover, the practical implementation of our behavior prediction solution should be tested with real users. The resulting surveys could shed light on further research opportunities.

REFERENCES

[1] E. Adar, D. S. Weld, B. N. Bershad, and S. S. Gribble, "Why we search: Visualizing and predicting user behavior," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 161–170. [Online]. Available: https://doi.org/10.1145/1242572.1242595

[2] P. W. Szabo, *User Experience Mapping*. Packt Publishing, 2017.

[3] A. Even, "Analytics: Turning data into management gold," *Applied Marketing Analytics*, vol. 4, no. 4, pp. 330–341, 2019.

[4] B. L. Velammal, "Typicality-based collaborative filtering for book recommendation," *Expert Systems*, vol. 36, no. 3, p. e12382, 2019.

[5] M. Patil and M. Rao, "Studying the contribution of machine learning and artificial intelligence in the interface design of e-commerce site," in *Smart intelligent computing and applications*. Springer, 2019, pp. 197–206.

[6] M. V. R. Senthilkumar, C. R. Kiron, M. Vishnuvarthan *et al.*, "A new approach to product recommendation systems," 2019.

[7] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*. Springer, 2007, pp. 291–324.

[8] J.-W. Choi, S.-K. Yun, and J.-B. Kim, "Improvement of data sparsity and scalability problems in collaborative filtering based recommendation systems," in *International Conference on Applied Computing and Information Technology*. Springer, 2019, pp. 17–31.

[9] P. Szabo and B. Genge, "Efficient conversion prediction in E-Commerce applications with unsupervised learning," in *The 28th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2020)*, Hvar, Croatia, sep 2020.

[10] M. Kechinov, "ecommerce purchase history from electronics store - january - november 2020," 11 2020, kaggle dataset, provided by the REES46 Marketing Platform. [Online]. Available: https://www.kaggle.com/mkechinov/ecommerce-purchase-history-from-electronics-store

[11] ——, "ecommerce events history in cosmetics shop - october 2019 - february 2020," 3 2020, kaggle dataset, provided by the REES46 Marketing Platform. [Online]. Available: https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop

[12] V. Moscato, A. Picariello, and G. Sperli, "An emotional recommender system for music," *IEEE Intelligent Systems*, 2020.

[13] P. W. Szabo and Z. L. Janosi, "Using machine learning and behavioural analysis for user-tailored viewer experience," in *Proceedings of the 2019 IBC Show*, ser. IBC2019, 2019.

[14] N. Hug, "Surprise: A python library for recommender systems," *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020.

[15] A. Beheshti, S. Yakhchi, S. Mousaeirad, S. M. Ghafari, S. R. Goluguri, and M. A. Edrisi, "Towards cognitive recommender systems," *Algorithms*, vol. 13, no. 8, p. 176, 2020.

[16] Z. Wang, X. Yu, N. Feng, and Z. Wang, "An improved collaborative movie recommendation system using computational intelligence," *Journal of Visual Languages & Computing*, vol. 25, no. 6, pp. 667–675, 2014.

[17] X. Zhao, "A study on e-commerce recommender system based on big data," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2019, pp. 222–226.

[18] M. K. Najafabadi, A. H. Mohamed, and M. N. Mahrin, "A survey on data mining techniques in recommender systems," *Soft Computing*, vol. 23, no. 2, pp. 627–654, 2019.

[19] A. Shin, D.-J. Shin, S. Cho, D. Y. Kim, E. Jeong, G.-I. Yu, and B.-G. Chun, "Stage-based hyper-parameter optimization for deep learning," *arXiv preprint arXiv:1911.10504*, 2019.

[20] Z. Cai, Y. Long, and L. Shao, "Classification complexity assessment for hyper-parameter optimization," *Pattern Recognition Letters*, vol. 125, pp. 396–403, 2019.

[21] M. F. Dacrema, P. Cremonesi, and D. Jannach, "Are we really making much progress? a worrying analysis of recent neural recommendation approaches," in *Proceedings of the 13th ACM Conference on Recommender Systems*, ser. RecSys '19. Association for Computing Machinery, 2019, p. 101–109.

[22] P. Szabo and B. Genge, "Hybrid hyper-parameter optimization for collaborative filtering," in *22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2020)*, Timisoara, Romania, sep 2020.

[23] Y. Shao and Y.-h. Xie, "Research on cold-start problem of collaborative filtering algorithm," in *Proceedings of the 2019 3rd International Conference on Big Data Research*, 2019, pp. 67–71.

[24] N. Silva, D. Carvalho, A. C. Pereira, F. Mourão, and L. Rocha, "The pure cold-start problem: A deep study about how to conquer first-time users in recommendations domains," *Information Systems*, vol. 80, pp. 1–12, 2019.

[25] Y. Zhu, J. Lin, S. He, B. Wang, Z. Guan, H. Liu, and D. Cai, "Addressing the item cold-start problem by attribute-driven active learning," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[26] S. Natarajan, S. Vairavasundaram, S. Natarajan, and A. H. Gandomi, "Resolving data sparsity and cold start problem in collaborative filtering recommender system using linked open data," *Expert Systems with Applications*, vol. 149, p. 113248, 2020.

[27] E. Castillejo, A. Almeida, and D. López-de Ipina, "Social network analysis applied to recommendation systems: Alleviating the cold-user problem," in *International Conference on Ubiquitous Computing and Ambient Intelligence*. Springer, 2012, pp. 306–313.

[28] M. Dong, F. Yuan, L. Yao, X. Xu, and L. Zhu, "Mamo: Memory-augmented meta-optimization for cold-start recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 688–697.

[29] S. Natarajan, S. Vairavasundaram, S. Natarajan, and A. H. Gandomi, "Resolving data sparsity and cold start problem in collaborative filtering recommender system using linked open data," *Expert Systems with Applications*, vol. 149, p. 113248, 2020.

[30] F. Zhang, S. Qi, Q. Liu, M. Mao, and A. Zeng, "Alleviating the data sparsity problem of recommender systems by clustering nodes in bipartite networks," *Expert Systems with Applications*, p. 113346, 2020.

[31] S. Yadav and S. Shukla, "Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification," in *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, Feb 2016, pp. 78–83.

[32] A. K. Nandi and H. Ahmed, *Classification Algorithm Validation*. IEEE, 2019, pp. 307–319. [Online]. Available: https://ieeexplore.ieee.org/document/8958927

[33] S. A. Vavasis, "On the complexity of nonnegative matrix factorization," *SIAM Journal on Optimization*, vol. 20, no. 3, pp. 1364–1377, 2010.

[34] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[36] H. Wang, D. Lian, and Y. Ge, "Binarized collaborative filtering with distilling graph convolutional networks," *arXiv preprint arXiv:1906.01829*, 2019.

[37] Y. Zhang, C. Yin, Q. Wu, Q. He, and H. Zhu, "Location-aware deep collaborative filtering for service recommendation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[38] N. Nassar, A. Jafar, and Y. Rahhal, "A novel deep multi-criteria collaborative filtering model for recommendation system," *Knowledge-Based Systems*, vol. 187, p. 104811, 2020.

[39] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[40] C. Wei, S. Kakade, and T. Ma, "The implicit and explicit regularization effects of dropout," 2020.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[42] W. Falcon *et al.*, "Pytorch lightning," https://github.com/PytorchLightning/pytorch-lightning, 2019.

[43] F. Pukelsheim, "The three sigma rule," *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/00031305.1994.10476030

[44] Q.-Y. Hu, Z.-L. Zhao, C.-D. Wang, and J.-H. Lai, "An item orientated recommendation algorithm from the multi-view perspective," *Neurocomputing*, vol. 269, pp. 261–272, 2017.

[45] J. McInerney *et al.*, "Explore, exploit, and explain: Personalizing explainable recommendations with bandits," in *Proceedings of the 12th ACM Conference on Recommender Systems*, ser. RecSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 31–39. [Online]. Available: https://doi.org/10.1145/3240323.3240354

Péter Szabó received the M.S. degree from "Petru Maior" University of Târgu Mureș, Romania, in 2008. He specialized in user experience and worked as the UX/UI department's senior manager at The Stars Group in London, UK. He is currently pursuing a Ph.D. degree in Informatics at the University of Medicine, Pharmacy, Science and Technology "G.E. Palade" of Târgu Mureș. His research interests are machine learning and using ML to enhance user experience.

Béla Genge is a Marie Curie Fellow and Professor at the University of Medicine, Pharmacy, Sciences and Technology (UMFST) of Targu Mures, Romania. He obtained his PhD in 2009 in network security from the Technical University of Cluj-Napoca, Romania, and acquired a 3-year experience as Post-Doctoral researcher at the Institute for the Protection and Security of the Citizen, Ispra, Italy (2010-2013). He has authored several papers in peer reviewed journals including IEEE Transactions on Emerging Topics in Computing, IEEE Transactions on Smart Grid, IEEE Communication Surveys and Tutorials, IEEE Systems, Communications of the ACM, International Journal of Critical Infrastructure Protection, Security and Communication Networks, Future Generation Computer Systems. He is a member of the editorial board of the International Journal of Critical Infrastructure Protection, Security and Communication Networks, Journal of Information Security and Mobility, and served as a TPC member for numerous international events including IEEE/IFIP Networking, ACM Workshop on Cyber-Physical Systems Security and PrivaCy, the Central European Cyber Security Conference, the International Symposium for ICS and SCADA Cyber Security Research. His research interests include security and resilience of in-vehicle communication systems, networked control systems, and security in the Industrial Internet of Things.