Ludwig-Maximilians-Universität München
Department of Statistics

Master's Thesis

What next? Modeling human behavior using smartphone usage data and (deep) recommender systems

Simon Wiegrebe

supervised by
Dr. David Rügamer

September 28, 2021

Contents

List of Figures

List of Tables

Abstract

Smartphone usage data has become an appealing source of data, not just to commercial entities but also to behavioral researchers. In this work, we take a data-driven approach to the exploration of behavioral sequences: we perform a variety of next-event prediction tasks on smartphone app sessions from a PhoneStudy project dataset, exploring many of the idiosyncrasies as well as different representations of our data. To do so, we employ a set of algorithms from the RS domain, well suited because of their ability to account for the sequential nature of app usage data and to deal with the large number of distinct events (e.g., apps) to be predicted.

We find that behavioral sequences can be modeled well by means of next-event prediction using RS methods. Many - especially non-deep - algorithms show strong predictive power across most tasks, partially achieving *HitRate@1* and *HitRate@20* performances of above 40% and 90%, respectively. And while deep RS methods work well on some tasks, they exhibit shortcomings caused by short sequence length and small dataset size.

1 Introduction

Recent years have seen smartphone usage data becoming a highly valuable source of data because of its large volume and ubiquitousness. While mostly employed by commercial enterprises - for example, for product targeting and personalized mobile marketing (Tong et al., 2020) - this data is also of increasing interest to researchers, due to its easy accessibility, cleanness, and high degree of representativeness of actual human behavior (Mafrur et al., 2015; Montag et al., 2019).

In this work, we seek to learn about human behavioral sequences through smartphone usage data. Yet whereas most behavioral research aims at associating observed smartphone usage patterns with self-disclosed behavioral traits (Peltonen et al., 2020; Stachl et al., 2020), in this work we aim to predict actual, observed behavior. We consider the PhoneStudy project data (Stachl et al., 2019), which contains smartphone app data for 310 participants over a period of approximately three months, and take an entirely data-driven approach: by construction, app usage data is sequential, i.e., time-ordered, and the number of distinct apps is very large, usually in the thousands. Sequence-aware models from the recommender systems (RS) domain naturally lend themselves to the analysis of this kind of data; and recently, deep learning-based RS approaches have further widened the repertoire of RS methods. In line with our exploratory approach, we choose a broad selection of eleven sequence-aware RS methods and perform various next-event prediction tasks. As explained in Section 3.2, we represent the app usage data - and thus the events to be predicted - in two distinct ways, applying our models to both "app-level" and "sequence-level" data.

We find that RS algorithms are capable of learning behavioral sequences, with multiple algorithms correctly predicting more than 30% of next apps (*HitRate@1*) and some algorithms reaching a 90% *HitRate@20* performance. We also find that neural network-based models are outperformed by simpler algorithms based on trees, co-occurrence frequencies, or nearest neighbors. As compared to standard RS datasets - as in Sun et al. (2019) or Latifi et al. (2021) - the predictive performance we obtain is impressive. This is partly due to the specific characteristics of our data: the event space - i.e., the variety of different apps - is not as large as in other RS tasks such as movie recommendation or next-click prediction, and some events occur very frequently, facilitating prediction. However, some of the data characteristics also pose limits in terms of modeling choices and performance: each app usage session commences by switching on the phone and ends by switching it off, implying that two events of each session are (almost) non-informative; furthermore, most sessions are rather short, which makes it hard to learn from them - in particular for neural network-based models. And while we explore alternative modeling approaches, such as removing ON and OFF events or representing the data from a sequence-level perspective, each one also comes with its own limitations. Altogether, we still conclude that behavioral sequences can be modeled successfully using RS methods, but also that the limitations of our data are likely a performance bottleneck.

This paper is organized as follows: In Section 2 we present the relevant theoretical background and approaches to the modeling of behavioral sequences. Section 3 describes our dataset and discusses different ways of how to represent (and subsequently model) the data. Section 4 introduces our selection of algorithms as well as the evaluation methodology. Section 5 presents our analyses along with the corresponding results. Section 6 concludes, addresses limitations, and provides suggestions for future research.

2 Theoretical Framework

Investigating human behavioral traits through smartphone usage data has been a topic of increasing interest for behavioral researchers and psychologists in recent years. Early explorations of this topic were made by Chittaranjan et al. (2013) and de Montjoye et al. (2013), who use features extracted from smartphone usage data to predict Big Five personality traits. More recently, Peltonen et al. (2020) gather smartphone application data over a period of six months as well as a questionnaire on Big Five personality traits for a total of 739 participants. They then use app (and app-category) usage to predict these personality traits. Stachl et al. (2020) use data from 743 participants recruited as part of the PhoneStudy project (Stachl et al., 2019) in order to predict Big Five personality dimensions based on behavioral information; this behavioral information is extracted from smartphone usage through a classification of activities (mobility, communication, etc.).

While datasets, extracted features, and prediction methodologies differ among the above-mentioned works, all of them aim to predict self-disclosed behavior, typically based on a questionnaire about pre-established personality traits. However, in order to figure out how people actually act - instead of how they proclaim they will be acting - it might be more instructive to try to directly predict people's actions. To do so, one can take advantage of the sequential nature of smartphone usage data, in particular of app usage, and focus on the prediction of the next app a user is going to use given their current usage session. However, none of the aforementioned works take up the notion of a usage session.

In this work, we also use the PhoneStudy project data (Stachl et al., 2019), yet we seek to explore human behavioral sequences from a data-centric point of view: we uncouple the data from its psychological context and simply consider apps as one-hot encoded app vectors, the same way word tokens are represented in a vocabulary in Natural Language Processing (NLP).

From there, one could project the one-hot encoded app vectors into a lower-dimensional embedding space, as done in Ma et al. (2016). The authors introduce *app2vec*, which adjusts the well-known *word2vec* (Mikolov et al., 2013) to the idiosyncrasies of app sequences; for instance, they modify the continuous bag of words (CBOW) algorithm to weight context words according to their temporal proximity to the target word. However, we do not choose this approach as our main modeling objective because of the following two main drawbacks: first,

while word analogies are very intuitive - like the famous "king - man + woman = queen" analogy by Mikolov et al. (2013) - app embeddings are much less intuitive and more difficult to construct. For instance, it is not very clear whether the app analogy examples given in Table V of Ma et al. (2016) are analogies at all. Second, app embeddings do not directly lend themselves to any type of standard performance evaluation: apart from app analogies or (rather hand-crafted) clustering tasks, there are no widely used performance metrics or methodologies available for app embeddings, making it hard to evaluate this modeling approach. Still, in section 5.1.3 we do experiment with app embeddings which are created internally by some of our selected algorithms.

Another approach to model behavioral traits, again disassociated from the psychological context, is to focus entirely on the sequential nature of the data. Our data is organized into time-ordered sequences - smartphone usage sessions - with a natural structure: they start by turning on the screen and end by turning it off. In its sequential nature, smartphone usage data is very similar to data from movie ratings, e-commerce sessions, or social networking sites: there are several users, each user has one or more sessions (depending on whether user ID is recorded), and each session contains one or more events (e.g., apps used or items clicked). For all of these data types, the typical task performed is next-event prediction - which in the case of smartphone usage data would be next-app prediction - and to do so, models from the recommender systems (RS) domain are used. RS methods are designed to be used whenever the target variable follows a multinomial distribution with a large number of distinct outcomes and the task is to recommend one (or more) of these outcomes. RS methods thus generate a list of top recommendations based on historical session data.

The sequential nature of our data is intrinsically similar to language data: events correspond to word tokens, while usage sessions correspond to sentences. Because of that, and due to the breakthroughs in NLP, language models have been adjusted to be used in the RS domain in recent years. These models are usually neural network-based and thus methodologically more sophisticated (see, e.g., *BERT4Rec* by Sun et al. (2019)), whereas many traditional RS methods are much simpler, based on co-occurrence frequencies or nearest neighbors (see baseline models in section 4.1.1). In order to model smartphone usage data, the RS methods used must be capable of handling the inherent sequential structure. Early work on next-app prediction - the task of predicting the next app in a session given all previous apps in that session - was carried out by Natarajan et al. (2013), who create a personalized recommender algorithm based on collaborative filtering and cluster-level Markov models. Another early work was conducted by Baeza-Yates et al. (2015), who model next-app prediction as classification task, using feature engineering and a Tree Augmented Naive Bayesian Network.[1] Since these early methods, literature on RS methods designed to take session data as input has grown considerably. In line with Latifi et al. (2021), sequential RS methods can be divided into two categories: session-based

[1]We do not include any of these two models into our analysis because the papers do not publicly share their code.

and session-aware models.

Session-based RS methods process session data as input, yet do not account for the potential existence of user-level clusters caused by multiple sessions stemming from the same user. That is, only recent events from the session at hand (short-term preferences) enter the model and thus generate predictions, whereas user-level effects (long-term preferences) do not. As a consequence, even if we have user-level information at hand (typically in the form of a user ID in addition to the session ID), session-based models cannot harness this information. In terms of past research, early work on session-based RS methods dates back to the beginning of this century (Mobasher et al., 2002; Shani et al., 2005); yet it was after the introduction of deep learning methods to the RS domain that many session-based models have been developed. The first deep learning-based session-based RS model was *GRU4Rec* (Hidasi et al., 2015), which is also the most well-known model of this category and is oftentimes used as baseline for both session-based (Li et al., 2017; Yuan et al., 2018) and session-aware approaches (Hu et al., 2018; Phuong et al., 2018; Quadrana et al., 2017; Ruocco et al., 2017). We describe the *GRU4Rec* model, as well as models that build on it, in more detail in Sections 4.1.2 and 4.1.3.

Session-aware RS methods, on the other hand, exhibit user-awareness; that is, they can account for both short-term session preferences (intra-session information) and long-term user preferences (inter-session information). In general, session-aware models have not been researched extensively as of now. An early (deep learning-based) session-aware RS method is *HGRU4Rec*, which is a user-aware extension of *GRU4Rec*; we explain the details of this model in Section 4.1.3.

In this work, we decide to focus on the session structure of our data and to investigate human behavior by performing next-event prediction, yet at the same time we go beyond mere app-level prediction (see Sections 3.2.2 and 5.2). The next section introduces our dataset as well as some necessary preprocessing.

3 Data

3.1 Description

We use the PhoneStudy dataset from a mobile sensing research project at the Ludwig-Maximilians-University Munich (Stachl et al., 2019).[2] As part of the study, 310 participants (users) were recruited, consenting to providing log, sensor, and app-usage data during the study period from October 29, 2017 through January 22, 2018.

For each user, we have up to 86 days of smartphone usage data, each app usage being recorded along with its exact opening date and time. In a first preprocessing step, we define

[2]The data is composed of three individual datasets. Combining all datasets would add heterogeneity in terms of participant recruitment as well as beginning, end, and duration of the study period. Moreover, the unbiased creation of windows (see Section 3.2) would become more difficult. Since the first dataset furthermore contains almost 95% of all observations, we decide to not include data from the second and third dataset.

smartphone usage sessions as a time-ordered sequence of apps (events), starting with an ON event (whereby the screen is switched on) and ending with an OFF event (whereby the screen is switched off); that is, ON and OFF are events just like any other app. All events within a single session are assigned the same unique session ID. By definition, each session has minimum length two; sessions with a length of exactly two are ON-OFF sessions, where a user turns on the screen (for example to check the time) and turns it off immediately after.

As a final general preprocessing step, we map app names to numeric app IDs and convert the date and time variable to a Unix timestamp, resulting in the dataframe as shown in Table 1. There is a total of 844,296 sessions, with 1392 and 1389 being the IDs corresponding to an ON and OFF event, respectively.

userID	timestamp	sessionID	appID
1	1.511423e+09	1	1392
1	1.511423e+09	1	1389
1	1.511424e+09	2	1392
1	1.511424e+09	2	1389
1	1.511424e+09	3	1392
...	...	...	...
310	1.515952e+09	844295	1389
310	1.515953e+09	844296	1392
310	1.515953e+09	844296	1389

Table 1: Excerpt of anonymized app-level data. The timestamp column contains Unix timestamps, i.e., seconds passed since January 01, 1970 (UTC).

3.2 Data Representation and Preprocessing

In language modeling there is a natural way of how to construe the data: tokens correspond to words (or word pieces), while a sentence is simply a concatenation of tokens ending with a period. In our case, the main objective is to model human behavior through smartphone usage data in an exploratory fashion, not to find a new use case for the application of language models. In the following, we therefore explore app-level and sequence-level representations, two distinct ways of how to construe and accordingly tokenize the data. We additionally discuss an app-to-text conversion approach, though do not further pursue it in this paper.

3.2.1 App-level Data

The simplest and most intuitive representation of our data is to let tokens correspond to apps and sentences correspond to sessions. As explained above, we define sessions to start with an ON event and end with an OFF event, whereby ON and OFF are treated as tokens just like any other app (event) token. We coin this "app-level data representation". The objective here is next-app prediction, i.e., predicting the next app (token) a user is going to use in a given

session (sentence). In our case, for each user we have a large number of sessions yet most of them are rather short, as can be seen in Table 2.

	App-level	Sequence-level
Number of events	4,314,830	720,379
Number of unique events	2,488	2,454
Number of sequences	844,296	9,377
Number of users	310	310
Sequences per user	2,723.54	30.25
Mean number of events per sequence	5.11	76.82
1st quartile of number of events per sequence	2.0	34.0
Median number of events per sequence	4.0	66.0
3rd quartile of number of events per sequence	6.0	106.0

Table 2: Summary statistics of app-level and sequence-level data.

As explained in Section 4.2.2, for performance evaluation all but the first event (app) of each test session is taken into account. We initially require all sessions to have a minimum length of two, as well as a minimum global frequency of five, in line with Latifi et al. (2021).[3] Furthermore, each user is required to have at least three sessions: one for training, validation, and testing, respectively.[4] Events from the validation and test set which do not occur in the respective user's training set are filtered out, as is explained in more detail in Section 4.2.1.

The short average sentence length of our app-level data is also a result of the high frequency of ON-OFF sessions: in fact, 328,554 out of all 844,296 sessions (38.9%) are such ON-OFF sessions. To increase average sentence length, in section 5.1.1 we remove the non-informative ON and OFF events (and thus implicitly all ON-OFF sessions).

Due to the large number of sessions per user and the nature of recommender systems, requiring all but the last session (of each user) for training, the test set is relatively small and only contains sessions from January 22, 2018 (the last day of the study period), which might introduce unwanted idiosyncrasies. To obtain more stable results, we split the entire study period - October 29, 2017 through January 22, 2018 - into five windows of equal length and assign sessions to windows based on the timestamp of a session's first app; this way, for each user we obtain five training, validation and test sets, respectively. Performance results are then simply averaged across the five test sets.[5]

[3]The minimum length requirement is not binding initially, since all of our sessions have minimum length two by construction, due to the presence of the ON and OFF event. In Section 5.1.1, we increase the minimum length requirement to focus on longer sessions only.

[4]This requirement does not exclude any of the users, with the average number of sessions per user being well above 2000.

[5]This multiple-window approach, in particular with the number of windows set to five, is a common procedure in the RS literature; see, e.g., Ludewig and Jannach (2018) or Latifi et al. (2021).

3.2.2 Sequence-level Data

While sessions for RS applications tend to be shorter than natural language sentences (see, e.g., Table 2 in Latifi et al. (2021)), our sessions are still particularly short. This is a potential concern for the use of neural network-based models, because they are used to learning from natural language sentences, which in turn tend to be significantly longer than our app-level sessions. After training, each new sentence constitutes a cold-start situation for the neural models, in particular for those without session awareness (Ludewig and Jannach, 2018).

One way to address the issue of having very short sessions is to view the data differently: instead of considering individual apps and their predictability, we might rather focus on the content of entire smartphone usage sessions and whether their content can predict the content of future sessions. To do so, in a first step we redefine an app session as concatenation of the *categories* of the apps which constitute a session, using pre-defined app categories coming with our dataset.[6] In doing so, we also combine consecutive events from the same category into a single event. This categorization reduces the number of unique sessions from 173,808 to 124,624. In a second step, we condense each (categorized) session into a single "content token" by concatenating the app categories within a session into a single session string. In a third step, for each user we create sentences by grouping together all "content tokens" occurring on a single calendar day; we do so by assigning a unique sentence ID to all of a given user's session strings where the timestamp of the session strings' first app occurs on a given calendar day. We choose calendar days for defining a sentence because of their natural cyclical pattern and because the resulting sentences have a much higher - yet not too high - average length as compared to app sessions (see Table 2 for summary statistics). This way, we again obtain a sequential data structure: for each user there are multiple sentences, which in turn contain multiple tokens. We call this sequence-level analysis.

Up until now, the term *session* has been used ambiguously, referring to a sequence of events in app-level analysis and to a single event token in sequence-level analysis. Furthermore, the term *sentence* has been utilized as sequence-level analog of an app-level session as well as in its original natural language meaning. To avoid confusion, from now on we mostly use the unambiguous terms *event* (to refer to a single token) and *sequence* (to refer to a sequence of tokens).

As for preprocessing of sequence-level data, we again require a minimum sequence length of two events, at least three sequences per user, and events from validation and test set to also appear in the training set. Now, however, the minimum global frequency for an event to be included is set to ten (as compared to five for app-level analysis); this removes many of the very infrequent events and we end up with 2,454 unique events, comparable to the app-level setting.

An evident drawback of sequence-level analysis is the data size. As can be seen in Table

[6] For instance, the app category corresponding to "WhatsApp" is "Messaging". This app-category mapping, which maps the 2,488 individual apps to a total of 93 categories, was performed by Stachl et al. (2020).

2, the number of total *events* is close to what used to be the number of *sequences* in the app-level setting: 720,379 versus 844,296, the difference being mostly due to the minimum global frequency of ten required for sequence-level events. The number of sequences is now significantly lower.

In sequence-level analysis, we now have ON-OFF events (which used to be ON-OFF sequences in the app-level setting), constituting 51.06% of all events. In section 5.2, we remove all ON- and OFF-events from the underlying (app-level) data before converting to the sequence-level representation, thereby also removing all ON-OFF tokens.

Finally, as explained for app-level data above, we conduct both single-window and multiple-window performance evaluation, with windows being created exactly as described in Section 3.2.1.

3.2.3 App-to-text Conversion

Yet another way to construe the data would be to start with the methodology we wish to use and then represent the data accordingly. In particular, we might choose to use transformers (Vaswani et al., 2017), given their breakthroughs in NLP in recent years and recalling that language data is, by definition, also sequential. Using transformers has one key advantage: one can make use of transfer learning, by pre-training the model on the vast amounts of task-agnostic text data available and then using the pre-trained model for task-specific fine-tuning. As for the latter step, there are two possible approaches in terms of how to represent our data for fine-tuning.

First, we could convert our app sequences into natural language by using rule-based natural language descriptions of each app. An app sequence would then correspond to a natural language sentence.[7] This approach has two advantages: first, it would allow for inclusion of further features beyond the mere app name; for instance, one could easily include an app's timestamp and geolocation into the app usage description. Second, the transformer is both pre-trained and fine-tuned on (sequential) language data, so there is no inefficiency or bias caused by different data modalities.

Alternatively, we could try to model our app sequences directly - and thus make next-app predictions directly - by representing our data as in section 3.2.1, i.e., without transcribing app sequences to natural language. The idea behind this approach, as explored in Lu et al. (2021), is that transformers do not only learn language sequence patterns but sequential patterns in general, even if data of different modalities is involved. This way, a transformer pre-trained on language data could directly be fine-tuned on non-language sequential data, in our case on (tokenized) app sequences. While this approach has the advantage of not requiring (rule-based) app-to-language transcriptions, it does not allow for the abovementioned inclusion of additional

[7]For example, the app sequence "ON, WhatsApp, Calendar, OFF" could be transcribed into the following natural language sentence: "The user switched on their screen in order to chat on WhatsApp, then checked their calendar, and finally switched off the screen."

app-related information. Furthermore, it is unclear how well transformers would perform on this particular cross-modal task.

In this work, we do not further pursue analysis based on app-to-text conversion and transformers, instead leaving it for future research.

4 Methodology

Throughout the last few years, almost all session-based and session-aware models proposed have been neural network-based, with newer models showing a tendency to be architecturally ever more complex. However, as demonstrated in Ludewig and Jannach (2018) for session-based models and in Latifi et al. (2021) for session-aware models, complex neural methods oftentimes fall short of much simpler methods when a fair performance comparison is conducted. Indeed, unfair comparisons and undue declarations of new "state of the art" models are not restricted to the RS domain (see Shwartz-Ziv and Armon (2021) for examples). Our methodology thus consists of two key aspects: choosing a modeling framework and enabling a fair performance comparison.

4.1 Modeling

In section 3.2 above we saw that preprocessing is very dataset-specific. The prediction task, on the other hand, is not: we wish to predict the next event, based on a sequence of past events. Whether events and sequences are defined in an app-level or sequence-level fashion, or whether we are dealing with a different RS dataset altogether (e.g., e-commerce data instead of smartphone usage data), is therefore irrelevant regarding the choice and implementation of algorithms. We simply need a set of algorithms which take sequential data, potentially clustered at a user level, as input and output prediction scores for each candidate event; these prediction scores can then be ranked to form a recommendation list, which in turn is used to evaluate the algorithm by comparing recommended events to observed events.

We use the framework as implemented by Latifi et al. (2021), who compare a variety of RS algorithms applied to retail, music, and social networking datasets.[8] We restrict ourselves to a subset of all implemented algorithms, based on their type, applicability to our task, and runtime. Altogether we select eight algorithms, covering both session-based and session-aware methods and different degrees of model complexity. Sections 4.1.1 through 4.1.3 explain the algorithms selected in more detail. Moreover, we equip three session-based algorithms with session-aware extensions, as explained in Section 4.1.4. Our selection thus comprises a total of eleven algorithms. These algorithms are then applied to the next-event prediction tasks in Section 5.

[8] Implementation by Latifi et al. (2021): `https://github.com/rn5l/session-rec/`. Like them, we perform all modeling, evaluation, and analysis tasks in Python.

Before describing the algorithms, we quickly introduce some notation. Let $s_{complete} := (s_1, s_2, \ldots, s_m)$ be a sequence of chronologically ordered events, with s_s being the last "known" event in the sequence; that is, $s := (s_1, s_2, \ldots, s_s)$ is our current (known) sequence and event s_{s+1} is the true next event, which we seek to predict. S_p denotes the set of all training sequences. Let i be the candidate event for which we wish to compute a score, given the current sequence s. Finally, $\mathbb{1}_A(a)$ denotes the indicator function, equaling 1 if $a \in A$ and 0 otherwise.

4.1.1 Session-based Baseline Models

We use the term *baseline* to refer to models which are not neural network-based. We include five baseline algorithms (as well as two extension-equipped ones) into our analysis. The simplest baseline algorithms are called *Association Rules (AR)* and *Sequential Rules (SR)*, both introduced by Ludewig and Jannach (2018). These algorithms calculate a score for a given candidate event by simply counting (or looking up) the number of intra-sequence co-occurrences of the current sequence's last known event and the candidate event among all training sequences. Since these methods only take into account the last event within a sequence when making a prediction, they exhibit very low computational complexity.

AR is a simplification of the mining technique proposed by Agrawal et al. (1993). Among all training sequences, this method simply counts how many times s_s and i appear together in a sequence; this count is then normalized by the number of *all* co-occurrences of s_s (i.e., with *any* other event) to derive the score of i. Formally, the *AR*-score for current sequence s and candidate event i is

$$score_{\mathrm{AR}}(i, s) = \frac{1}{\sum_{p \in S_p} \sum_{x=1}^{|p|} \mathbb{1}_{\{s_s\}}(p_x) \cdot (|p| - 1)} \sum_{p \in S_p} \sum_{x=1}^{|p|} \sum_{y=1}^{|p|} \mathbb{1}_{\{s_s\}}(p_x) \cdot \mathbb{1}_{\{i\}}(p_y). \quad (1)$$

SR, originally proposed by Kamehkhosh et al. (2017), is a modification of *AR*. It takes into account the sequential order of events in two ways: by only considering co-occurrences where s_s *precedes* i and by decreasing the weight assigned to i if other events occurred between s_s and i. Formally, the *SR*-score for s and i is

$$score_{\mathrm{SR}}(i, s) = \frac{1}{\sum_{p \in S_p} \sum_{x=2}^{|p|} \mathbb{1}_{\{s_s\}}(p_x) \cdot (x)} \sum_{p \in S_p} \sum_{x=2}^{|p|} \sum_{y=1}^{x-1} \mathbb{1}_{\{s_s\}}(p_y) \cdot \mathbb{1}_{\{i\}}(p_x) \cdot w_{\mathrm{SR}(x-y)}, \quad (2)$$

with weighting function $w_{\mathrm{SR}}(z) := 1/z$. (See Ludewig and Jannach (2018) for further details.)

An entirely different approach is the *Context Tree (CT)*, proposed by Mi and Faltings (2018). It uses a variable-order Markov model with context-dependent depth of the Markov chain, circumventing the difficult choice of a single depth when using fixed-order Markov processes. *CT* builds a tree where each node is a context, defined to be the set of all sequences with a specific suffix (combination of last events), and where the edges combine parent and child

nodes (i.e., shorter and longer suffixes). For each context, an expert is then defined to be the probability of the next event given the current sequence of events, calculated using a Dirichlet-multinomial prior as proposed in Dimitrakakis (2010). The current sequence activates a path of nodes from the context tree and, as a consequence, all corresponding experts. Finally, all activated experts generate predictions which are, in turn, combined recursively to yield a final event prediction. We refer the reader to Dimitrakakis (2010) and to Section 3 of Mi and Faltings (2018) for further details on the algorithm and the generation of predictions.

The remaining three non-neural algorithms are nearest neighbor-based, with *Session-based kNN (SKNN)* being the base method and the *Sequence and Time Aware Neighborhood (STAN)* as well as the *VSTAN* methods being extensions thereof.

SKNN, originally proposed in Jannach and Ludewig (2017), distinguishes itself from *AR* and *SR* by not only focusing on the last (known) event of a sequence; instead, this method determines the similarity between the entire current sequence and past sequences. To be precise, one initially defines the neighbors of current sequence s, N_s, as the k past sequences with the highest similarity to s; similarity is measured in terms of a similarity function $sim(\cdot, \cdot)$, such as cosine similarity, applied to binary vectors over the event space, as done in Bonnin and Jannach (2014). k and $sim(\cdot, \cdot)$ are hyperparameters. Finally, the score for s and candidate event i is defined as

$$score_{\text{SKNN}}(i, s) = \sum_{p \in N_s} sim(s, n) \cdot \mathbb{1}_{\{n\}}(i), \tag{3}$$

i.e., simply as the sum of similarity scores across all sequences which contain candidate event i.

STAN, proposed by Garg et al. (2019), extends *SKNN* in three ways. First, it accounts for event recency in s: potential neighbor sequences are still binary vectors while event weights within s decrease with decreasing recency. To be precise, *STAN* applies an exponential (decay) function to all but the last event of s. This, in turn, translates into decreased contributions to the similarity score by less recent events if using similarity functions such as cosine similarity or the dot product. Second, while *SKNN* only considers sequence similarity in determining the set of neighbor sequences, *STAN* additionally takes into account the recency of other sequences with respect to s, again using an exponential decay factor to decrease the weight of sequences farther away from s. Third, *STAN* also considers the distance of i to other, shared events in N_s, with i being assigned an (exponentially) higher weight if it is closer to some other recommendable event i^*. The final score, $score_{\text{STAN}}(i, s)$, is then simply computed as for *SKNN*, yet augmented by these three weights.

Finally, *VSTAN* (Ludewig et al., 2021) combines the *SKNN*-modifications from *STAN* and *VSKNN*, another *SKNN*-extension proposed by Ludewig and Jannach (2018).

4.1.2 Session-based Neural Models

In terms of neural network-based session-based models we include the *GRU4Rec* model (Hidasi et al., 2015) as well as one extension, see Section 4.1.4. *GRU4Rec*, being the earliest deep learning- and session-based RS method, employs a Gated Recurrent Unit (GRU; Cho et al., 2014), a Recurrent Neural Network (RNN) architecture and standard choice for modeling sequential data such as time series, to generate predictions. The general model architecture is shown in Figure 1.

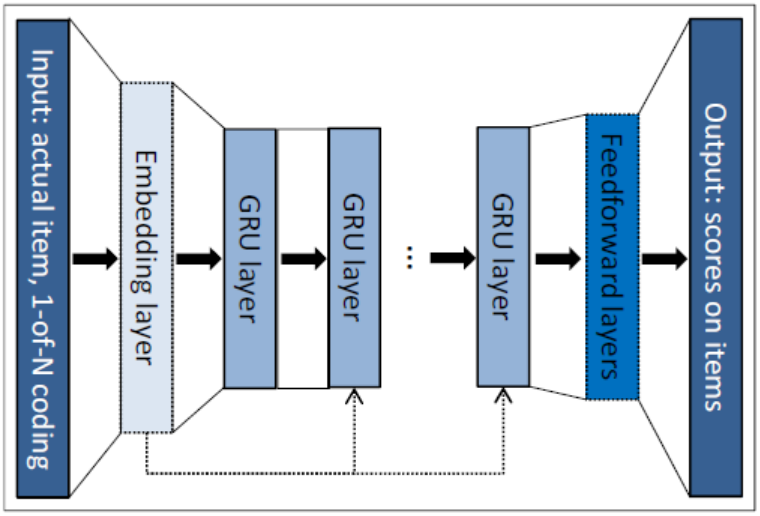

Figure 1: General architecture of the *GRU4Rec* model. Figure is taken from Hidasi et al. (2015).

GRU4Rec commences by one-hot encoding single input events into vectors of length equal to the event space. After an optional embedding layer (see also our experiments in Section 5.1.3), the input vector is fed into a GRU layer, followed by additional, optional GRU and feedforward layers.[9] For each distinct event, the model outputs the likelihood of being next in the sequence, which can be interpreted as a score.

Since the sequential data used in RS tasks is different from the primary applications of RNNs, the authors experiment with pointwise and pairwise ranking loss functions to be used for model training, concluding that only pairwise ranking loss functions work well. Pairwise ranking losses generally compare the score of the actually observed event s_{s+1} with the score of negative samples. Specifically, the authors suggest the following two loss functions: first, the Bayesian Personalized Ranking loss (*BPR*; Rendle et al., 2012), which is the negative logarithmized probability of the score of s_{s+1} exceeding the score of a negative sample. Second, their self-devised *TOP1* loss, which approximates the relative rank of the score of s_{s+1}, adding a

[9] From their experiments, the authors of *GRU4Rec* conclude that simple one-hot encoding with no additional embedding layers, a single GRU layer, and no feedforward layers yields the best performance.

regularization term to decrease negative sample scores. In a more recent version of their model (Hidasi and Karatzoglou, 2018), which is also the version we use here, the authors further improve the losses by making them focus on the highest-scoring sample (instead of all samples). This helps reduce the formerly large proportion of negative samples which produce gradients close to zero and thus cause the gradients to vanish. The loss function - either *BPR-max* or *TOP1-max* - is a hyperparameter to be tuned. *GRU4Rec* uses RMSprop (Hinton et al., 2012) for optimization.

Given the peculiarities of RS data and the loss functions, the authors make two additional modifications to the base network to reduce runtime and computational complexity: first, session-parallel mini batches are employed to speed up training; and second, in each step scores are only calculated for a subsample of all potential events.

4.1.3 Session-aware Neural Models

Finally, we include one session-aware method based on deep neural networks: *HGRU4Rec* (Quadrana et al., 2017) is a user-aware extension of *GRU4Rec*, containing a short-term and long-term memory component in the form of different GRU layers. To be precise, the model uses what the authors call a session-level GRU, which generates recommendations for each event in a sequence. At the end of each sequence, an additional user-level GRU is updated and its hidden state is then employed for initialization of the session-level GRU at the beginning of the next sequence. The architecture of *HGRU4Rec* is visualized in Figure 2. From this, it becomes clear that the basic mechanism of how a score is produced by *HGRU4Rec* is identical to *GRU4Rec* - the only difference is the initialization of the GRU at the beginning of a sequence. Note, however, that *HGRU4Rec* uses the original *TOP1* loss instead of the improved *TOP1-max* version. The model is optimized using AdaGrad (Duchi et al., 2011).

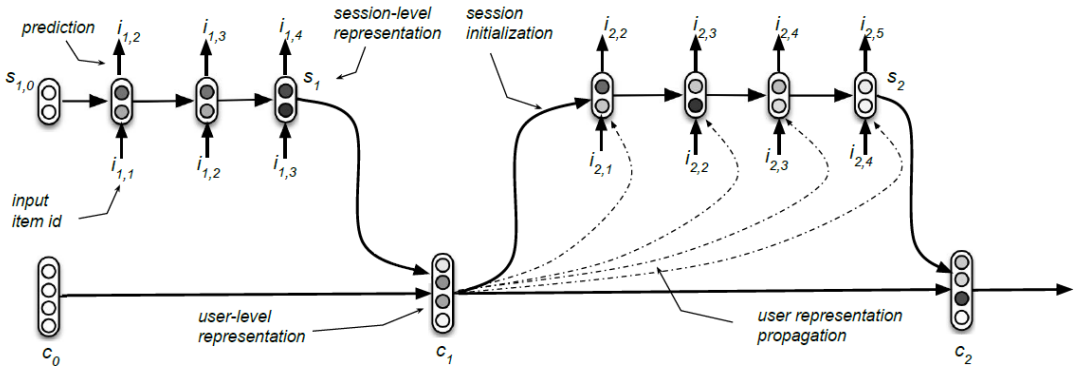

Figure 2: General architecture of the *HGRU4Rec* model. Figure is taken from Quadrana et al. (2017).

4.1.4 Extensions

A heuristic extension to a session-based model is an extension to the original algorithm providing user-level information from past sequences and thus making the respective algorithms session-aware. We use the following three heuristic extensions:[10]

> *Extend*: In case the current sequence length $|s|$ is lower than hyperparameter d, the most recent $d - |s|$ events from the user's preceding sequence(s) are simply prepended to the current sequence so that it has length d.
>
> *Boost*: This extension requires an algorithm to return scores. If so, it increases the score of each candidate event i by b percent (with b being a hyperparameter) whenever i has occurred at least once within the respective user's past sequences.
>
> *Remind*: This extension was originally proposed in Jannach et al. (2017). Its key idea is to create a reminder list containing events from the last p sequences of a user (with p being a hyperparameter), based on which a reminder score is calculated as a weighted combination of *interaction recency* and *session similarity* (see Jannach et al. (2017) and Latifi et al. (2021) for further detail). This reminder score is then added to the original algorithm's score for each recommended event.

We include three extension-equipped algorithms into our analysis. The first one is *SR_BR*, obtained by applying the *Boost* and *Remind* extensions to *SR*. (The *Extend* heuristic is useless here as *SR* only takes s_s, the last event in the current sequence, into consideration for prediction. Hence, making the current sequence longer does not impact prediction.) The remaining two algorithms are *VSTAN_EBR*, the *VSTAN* algorithm equipped with all three extensions, as well as *GRU4Rec_R*, which is the reminder-equipped *GRU4Rec* algorithm. Thus, our selection encompasses a total of eleven algorithms.

4.2 Evaluation and Tuning

We now proceed to discuss our evaluation and tuning methodology. It includes data splitting, the evaluation protocol and tuning strategy for our data, as well as the choice of appropriate evaluation and tuning metrics.

4.2.1 Train-Validation-Test Split

Due to the presence of sequence data and user IDs, our dataset is time-ordered and user-clustered, requiring a time- and user-aware data splitting procedure as compared to standard time-agnostic cross-validation. That is, we must make sure that for every user for whom we have a test sequence (i.e., a sequence of events in the test set), there are corresponding training

[10] Both design and implementation of these extensions are taken from Latifi et al. (2021).

sequences - and vice versa. Because of the sequential nature of the data, we use each user's last sequence as test set, in line with the literature (Guo et al., 2020; Latifi et al., 2021; Phuong et al., 2018, 2019; Quadrana et al., 2017). Using an earlier sequence as test set would imply using a user's *future* behavioral sequences for model training - which conflicts with our overall objective of *predicting* future behavior.

In standard cross-validation, the same data is split into k folds, with each fold being used as test set once and as part of the training set the remaining $k - 1$ times. This presupposes that examples can simply be randomly shuffled around and assigned to the training or test set. Clearly, this is not the case here: first of all, individual events are grouped into sequences and this sequence-ordering must always be maintained; that is, we cannot assign some events from within a single sequence to the training set and others to the test set. Second, since our sequences are user-clustered and some of our models are session-aware (i.e., they actively use this user-level information), neither can we randomly assign some sequences from a single user to the training set and others to the test set - because we must always have all but one user sequence in the training set and the remaining one in the test set. And finally, because of our user-level last-event split methodology, for a given user this test sequence must always be the last one - and, consequently, the training set must always comprise all but the last one.

Because of the above, standard cross-validation is not applicable here and, as a consequence, neither is nested cross-validation. In order to still be able to do hyperparameter optimization without introducing bias by tuning and evaluating on the same test data twice, we repeat the last-event split method twice: first, we apply it to the entire data, which creates a test set by "clipping off" each user's last sequence. Then we apply it to the remaining data, creating a validation set comprising each user's last sequence (of the non-test data) as well as a training set containing each user's remaining sequences. Thus, starting from a given user's sequences, the last sequence is the test set, the second to last sequence constitutes the validation set, and all remaining (previous) sequences form the training set. As already stated in section 3.2, it is now clear why this procedure requires every user to have at least three sequences - one for training, validation, and test set, respectively. The reason we remove events which do not appear in the training set is because our models cannot predict events they have not seen during training.

As briefly explained in section 3.2, in addition to single-window analysis (performing the train-validation-test split on the entirety of sequences for each user) we also conduct multiple-windows analysis by splitting the study period into five equally long sub-periods (windows). In this case, we perform the train-validation-test split as explained above, yet on each window separately.

4.2.2 Evaluation Protocol

In terms of evaluation of a single test sequence, we iterate through all of its events and iteratively compare the respective prediction - available for all but the first event - with the corresponding

ground truth. This approach, as done in Hidasi et al. (2015) and Latifi et al. (2021), is preferable to the one in Sun et al. (2019), where the authors only predict the *last* event of each test sequence, because here we harness all but one test sequence event - instead of only a single one. To carry out this evaluation strategy, however, we first need to decide on a definition of ground truth. We might define a set of ground truths for a sequence by treating all but the first event as ground truths for all evaluation positions, as done by Ludewig and Jannach (2018). However, since we are mostly interested in predicting the single next action of a user, we opt for a narrower definition: we define the ground truth for an evaluation position within the test sequence to be only the actually observed event at that specific position, i.e., only s_{s+1}.

4.2.3 Evaluation Metrics

Given the above evaluation protocol, we need to choose appropriate evaluation metrics. Recall that in the RS domain the target variable follows a multinomial distribution with a large number of categories and we wish to quantify the goodness of our recommendation list of length k, generated based on the highest-scoring predictions of our algorithms. Furthermore recall that we wish to perform next-event prediction and thus only consider the actually occurring event as ground truth. Therefore, we identify Hit Rate and Mean Reciprocal Rank as most appropriate performance metrics. Coverage and Popularity Bias are additionally included, though only for $k = 20$, to indicate the algorithms' degree of recommendation diversification. The metrics we consider are thus:

Hit Rate (HR): The Hit Rate of a recommendation list of length k ($HR@k$) is the most intuitive and simple RS metric. Given a total of n events to be predicted and evaluated, $HR@k$ is simply the fraction of events for which the corresponding recommendation list of length k includes the ground truth - regardless of the position within the recommendation list. Formally,

$$HR@k = \frac{\sum_{i=1}^{n} \mathbb{1}_{rl(k)_i}(y_i)}{n},$$

where y_i is the ground truth of the i-th event to be predicted and $rl(k)_i$ is the corresponding predicted recommendation list of length k. It follows that a *high* $HR@k$ is desirable. We consider $HR@1$, $HR@5$, $HR@10$, and $HR@20$ in order to cover a broad range of reasonable recommendation list lengths.

Mean Reciprocal Rank (MRR): The Mean Reciprocal Rank of a recommendation list of length k ($MRR@k$) is similar to $HR@k$ in that it also considers how many recommendation lists include the respective ground truths, yet $MRR@k$ additionally accounts for the ranking within the list. To do so, and in order for lists with highly ranked ground truth to have high $MRR@k$, this metric uses the reciprocal rank of the ground truth within the

recommendation list and averages this reciprocal rank across all n events. Formally,

$$MRR@k = \frac{\sum_{i=1}^{n} rr_i}{n},$$

where rr_i is the reciprocal rank of the i-th event, defined as the reciprocal of the rank of y_i within recommendation list $rl(k)_i$ if $y_i \in rl(k)_i$, and zero otherwise.[11] As with $HR@k$, a *high* $MRR@k$ is preferable. We consider $MRR@5$, $MRR@10$, and $MRR@20$.

Coverage: The Coverage of an algorithm with respect to a recommendation list of length k ($Coverage@k$) is the percentage of events (out of all possible events) that appear at least once in the recommendation lists. This way, this metric indicates the degree of personalization of an algorithm. We consider $Coverage@20$ as this reflects the coverage not only of the top recommendation but of a reasonably long recommendation list for each algorithm.

Popularity Bias: The Popularity Bias of an algorithm with respect to a recommendation list of length k ($Popularity@k$) is the degree to which an algorithm focuses only on the most popular events in making recommendations, thereby neglecting the long tail of not-so-popular potential events. $Popularity@k$ is calculated as average across popularity scores of each individual recommended event. For each individual recommended event, this score is simply the min-max normalized count of an event's appearances in the training sequences (see also Ludewig and Jannach (2018)). We consider $Popularity@20$, for the same reasons as with coverage.

In case of multiple-window analysis, we average the $HR@k$ and $MRR@k$ performances, respectively, across all five windows. For tuning we use $HR@1$, for reasons explained below.

4.2.4 Tuning

As for hyperparameter optimization, we choose a simple random search tuning strategy with budget equal to 100 for all algorithms. For our selected algorithms, we use the same hyperparameter search space as given in Tables A.11 and A.12 of Latifi et al. (2021).[12] *AR* and *CT* do not have any hyperparameters and thus do not need to be tuned.

Before explaining the actual tuning strategy, we first define the tuning data. We decide to tune on the unrestricted data, thus without excluding *ON* and *OFF* events. While excluding them would speed up tuning by decreasing dataset size significantly (recall that for app-level analysis, more then one-third of all sequences are *ON-OFF* sequences), the remaining

[11] By definition, it follows that $MRR@k \leq HR@k$ for all $k \in \mathbb{N}$, with equality for $k = 1$, which is the special case of making only a single recommendation per evaluation event.

[12] Since *SKNN* is not included in Latifi et al. (2021) but its hyperparameters are a subset of those from *VSKNN* (which is included), we simply use the *VSKNN* search space for all hyperparameters the two models have in common.

sequences would not be representative of all behavioral sequences anymore and we would thus lose generalizability.

Due to the data structure and our last-event splitting method, we do not use any cross-validation; instead, for each user there is a single validation sequence used for evaluation of the (up to) 100 different hyperparameter configurations per algorithm - and, as a consequence, we obtain a single optimal hyperparameter configuration per algorithm.

Conducting the tuning procedure on the entire single-window data would imply extremely long runtimes as well as a time bias (recall that the validation set of the single-window analysis would essentially coincide with the validation set of the last window), while tuning on only one window would merely address the former of these two issues. This is why we choose to perform hyperparameter tuning on all five windows for both app-level and sequence-level analysis - unlike Latifi et al. (2021), who only use the window which contains the largest number of events. Thus, for a given algorithm and hyperparameter configuration, we compute validation set performance - in terms of $HR@1$, as explained below - for each of the five windows separately and then average performance across windows. Subsequently, we choose the one configuration with the best average performance as optimal hyperparameter configuration. Tables 11 and 12 in the appendix contain the optimal hyperparameters for each selected algorithm for both app- and sequence-level data.

Finally, we need to choose a performance metric. Since we are mostly interested in predicting the single next event (i.e., a recommendation list of length $k = 1$), we choose $HR@1$ as the natural candidate. And while $HR@1$ performance tends to fluctuate significantly across windows, we account for this by using the aforementioned five-window tuning strategy.

After tuning and extracting the optimal hyperparameter configuration for each algorithm, we then (re-)train all algorithms on the combined train and validation set (i.e., for each user, on all sequences except the last one) and evaluate algorithm performance on the test set (for each user, the very last sequence).

5 Numerical Experiments

In this section, we apply our selection of eleven algorithms to our smartphone usage data. Sections 5.1 and 5.2 present app- and sequence-level results, respectively. In both sections, we go beyond a single next-event prediction task, conducting analyses to explore idiosyncrasies of both data and algorithms.

5.1 App-level Results

For app-level analysis, we first present overall results of the next-app prediction task, followed by an analysis of the effect of minimum sequence length, of the position within the test sequence, and of removing ON and OFF events. In the second subsection we consider prediction at

category level, while in the third subsection we add an additional embedding layer to the (tuned) *GRU4Rec* model and explore its semantic meaningfulness.

5.1.1 Next-event Prediction

Overall Performance

The main task of this paper is to explore predictability of future actions based on a sequence of past events. In the app-level setting, this equates to next-app prediction. Table 3 presents the multiple-window performance of our fully tuned algorithms where performance is averaged across all five windows. Figure 3 shows averaged five-window $HR@k$ performance for all algorithms and different values of k.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.2024	0.5926	0.7250	0.8407	0.3374	0.3555	0.3637	0.3208	0.2328
SR	0.2591	0.6431	0.7836	0.8568	0.4056	0.4248	0.4300	0.3552	0.1892
SR_BR	0.2718	0.6902	0.8203	0.8559	0.4314	0.4496	0.4523	0.3578	0.1892
CT	**0.3833**	**0.7006**	0.8193	0.8840	**0.5130**	**0.5290**	**0.5336**	0.3267	0.2491
SKNN	0.0159	0.5290	0.6876	0.7697	0.1909	0.2119	0.2178	0.0489	0.2463
STAN	0.1543	0.3386	0.3451	0.3459	0.2307	0.2317	0.2317	0.1395	0.0883
VSTAN	0.1543	0.5635	0.6769	0.7202	0.3086	0.3244	0.3276	0.3654	0.1838
VSTAN_EBR	0.2436	0.6462	**0.8290**	**0.9278**	0.3822	0.4077	0.4150	0.5576	0.2130
GRU4Rec	0.3190	0.5829	0.6701	0.7503	0.4218	0.4338	0.4394	0.8002	0.0895
GRU4Rec_R	0.3105	0.6280	0.7241	0.7949	0.4285	0.4417	0.4467	0.8200	0.1438
HGRU4Rec	0.2978	0.4247	0.4706	0.5182	0.3486	0.3548	0.3580	0.6824	0.0249

Table 3: Performance results on five-window app-level data. Performance is averaged across all windows, which are of equal time span. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

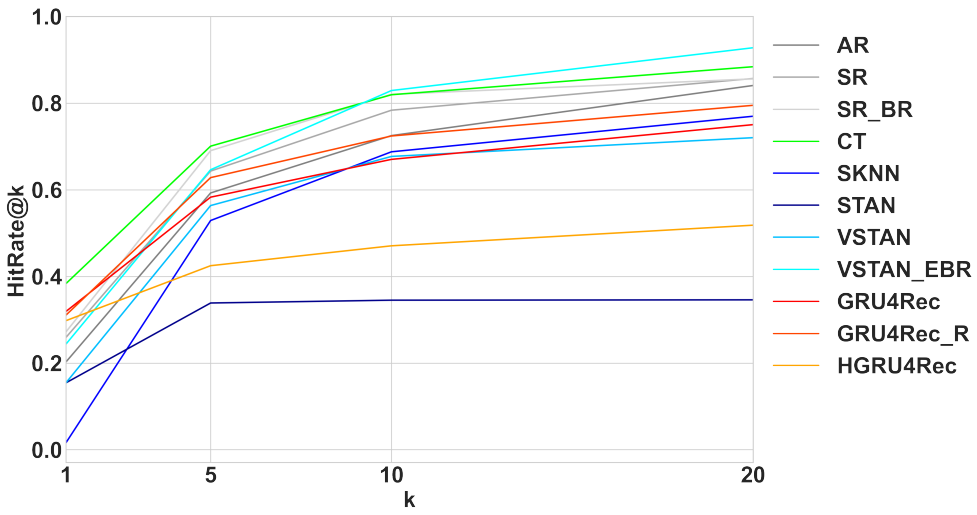

Figure 3: $HR@k$ performance for $k = 1, 5, 10,$ and 20 on five-window app-level data for all selected algorithms.

The *CT* algorithm achieves the best (i.e., highest) $HR@1$ and $HR@5$ performance, while

VSTAN_EBR performs best in terms of $HR@10$ and $HR@20$, i.e., when the Hit Rate is evaluated with regard to long recommendation lists. The fact that *CT* also has the highest $MRR@k$ (for all k) furthermore indicates that it not only produces the best top-k recommendations for low k, but that it also achieves the best (inverse) rank-weighted recommendation quality for longer recommendation lists. (This is consistent with *CT*'s strong $HR@1$ and $HR@5$ performance.) The three neural network-based models *GRU4Rec*, *GRU4Rec_R*, and *HGRU4Rec* come right behind *CT* when considering $HR@1$, yet their relative performance deteriorates as the recommendation list size increases. Amongst them, *HGRU4Rec* is, by large, the weakest performer, even more so for larger k. The *Remind* extension of *GRU4Rec* increases performance for $k \geq 5$. Finally, the *Coverage*@20 and *Popularity*@20 metrics indicate that the extensions increase an algorithm's app coverage, while they do not greatly affect its popularity bias. Neural network-based models have, by far, the highest app coverages and the lowest popularity biases, implying that the events they predict cover a broader range of the entire set of events as compared to other algorithms. Finally, we observe that session-based algorithms benefit from heuristic session-aware extensions.

For single-window analysis, no separate tuning is required because the data stems from the same population as the multiple-window data and, by construction, the test set of the single-window data is identical to the test set of the last window in the multiple-window analysis. The results, as well as performance differences vis-à-vis Table 3, are shown in the appendix (Tables 13 and 14). Apart from some discrepancies in relative performance for individual metrics, the results are very consistent with the multiple-window analysis; in particular, *CT* is the overall top performer for all metrics except $HR@10$ and $HR@20$.

Minimum Sequence Length

All neural network-based methods considered in this paper employ RNNs, which rely entirely on learning from the present sequence instead of "looking up" similar sequences or app co-occurrences, as is done by many non-neural methods. As discussed in Section 3.2, app-level sequences are typically very short, implying that RNN-based methods do not have "much to learn from" for many sequences. We therefore expect *GRU4Rec*, *Gru4Rec_R*, and *HGRU4Rec* to perform better on longer sequences, where the underlying RNNs are provided more events to learn from. For models based on co-occurrence frequencies - *AR*, *SR*, and *SR_BR* - we expect sequence length to have no impact on performance since only the last event of a sequence is actively used for prediction.

In order to investigate this, we first create a subset of our data by only considering sequences with at least 20 events. $HR@k$ results when training and evaluating our models on this subset of data, compared to the all-sequences results in Table 3, are displayed in Figure 4; results for all performance metrics are shown in the appendix (Table 15). Overall, *CT* is still the best-performing algorithm in terms of $HR@1$ and $MRR@k$ (for all values of k we considered).

As expected, performance of the co-occurrence frequency-based models is very similar to the all-sequences performance. The three neural network-based models are still the best performers behind *CT* and, indeed, their performance improved. This confirms our initial expectation that RNN-based models benefit from longer sequences. As for the neighborhood-based models, we observe moderate performance decreases for all algorithms and metrics, except for *STAN* evaluated in terms of $HR@1$. This implies that finding similar sequences works well even if - or, rather, especially if - the sequences under consideration are not overly long.

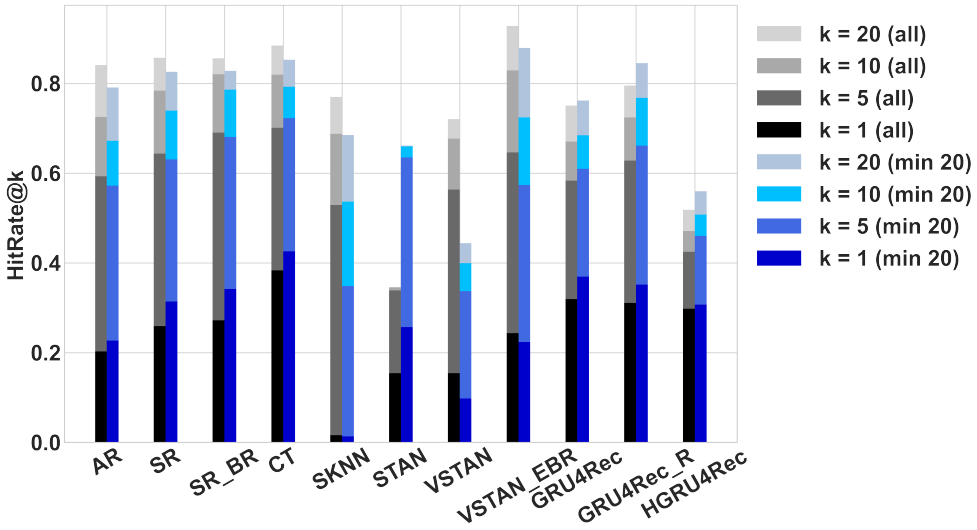

Figure 4: $HR@k$ comparison between performance on full five-window app-level data (left bars) and performance on five-window app-level data when only training and evaluating on sequences with a minimum length of 20, for $k \in \{1, 5, 10, 20\}$.

In a second step, we compare the above results to those obtained if we use all sequences for training and only apply the minimum length requirement to the sequences in the test set. That is, we use the same training data as in the initial multiple-window app-level analysis, yet we filter out all test sequences which contain fewer than 20 events. In Table 4 we present performance differences between the "long evaluation sequences only" setting and the "long training and evaluation sequences" setting; absolute performance results are reported in the appendix (Table 16).

The performance of all algorithms, except *VSTAN*, is now considerably worse. *CT* is still the best performer in terms of $HR@1$ and $MRR@k$ (for all k considered). We conclude that performance on long sequences decreases when using all sequences for training; or putting it differently: performance on long sequences benefits from training exclusively on long sequences. This is somehow surprising as the full training dataset is considerably larger than the long-sequences-only dataset and, usually, decreasing the training set size leads to weaker performance.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	-0.0375	-0.1070	-0.0800	-0.0010	-0.0684	-0.0649	-0.0594	-0.4834	0.0509
SR	-0.1330	-0.1556	-0.0744	-0.0269	-0.1405	-0.1306	-0.1272	-0.4782	0.0214
SR_BR	-0.1516	-0.1621	-0.0501	-0.0306	-0.1538	-0.1388	-0.1374	-0.5055	0.0165
CT	-0.1134	-0.1314	-0.0611	-0.0031	-0.1222	-0.1134	-0.1088	-0.4913	0.0412
SKNN	-0.0088	-0.0518	0.0367	-0.0416	-0.0326	-0.0239	-0.0293	-0.0225	0.0392
STAN	-0.0130	-0.0237	-0.0265	-0.0280	-0.0177	-0.0181	-0.0182	-0.2578	0.0252
VSTAN	0.1330	0.2688	0.3507	0.3740	0.1967	0.2077	0.2094	-0.2872	0.1637
VSTAN_EBR	-0.0887	-0.1345	-0.0218	0.0228	-0.1081	-0.0925	-0.0889	-0.5654	0.0846
GRU4Rec	-0.0847	-0.1080	-0.0966	-0.0818	-0.0968	-0.0955	-0.0946	-0.4507	-0.0133
GRU4Rec_R	-0.1425	-0.2068	-0.1994	-0.1756	-0.1710	-0.1701	-0.1687	-0.5142	0.0163
HGRU4Rec	-0.0351	-0.0506	-0.0377	-0.0370	-0.0415	-0.0399	-0.0399	-0.2834	-0.0059

Table 4: Difference between performance when training on all sequences but only considering sequences of at least 20 events for evaluation and performance when only considering sequences of at least 20 events for both training and evaluation on five-window app-level data.

Position in Test Sequence

Above we looked at minimum sequence length as a potential determinant of predictive performance since some models benefit from longer sequences. An alternative analysis regarding the same underlying issue - short sequences - is to look at performance by position within the test sequence. To do so, we do not aggregate performance across all positions and test sequences anymore; instead, we focus on a single position to be predicted and then average performance for that position across all test sequences.

Figure 5 shows performance in terms of $HR@1$ for the first ten prediction positions (which would be the second through eleventh positions in the original test sequences, recalling that no prediction is made for the first test sequence event); Table 17 in the appendix contains the underlying numbers. *CT* as well as the three neural models do not show any strong increase or decrease across positions, with the exception of a surprisingly strong performance on prediction position two. The results further indicate that the extension-equipped neighborhood-based model *VSTAN_EBR* performs very well on early positions but cannot keep up this performance for longer test sequences; *STAN* and *VSTAN*, on the other hand, perform poorly on the very first positions and then improve, reaching a similar performance as *VSTAN_EBR* roughly after the fifth position. A potential explanation is that *STAN* and *VSTAN* do not have any user-level heuristics and, additionally, the first event of all app-level sequences is a (non-informative) ON event, so for the first prediction positions it is very hard for these models to find similar sequences based on such unspecific information. The extensions of *VSTAN_EBR* appear to initially (more than) compensate for the low informational content of the current sequence, with this effect wearing off over the first five positions.

From the individual positions, it is sometimes hard to identify a pattern when the actual question is whether an algorithm performs better on "early" or "late" positions; furthermore, positions after the tenth position are not considered at all. Therefore, for each test sequence we introduce a cutoff after the second, fifth, or tenth event to be predicted (which would be the third, sixth, or eleventh event in the original test sequence) and subsequently compare pre-

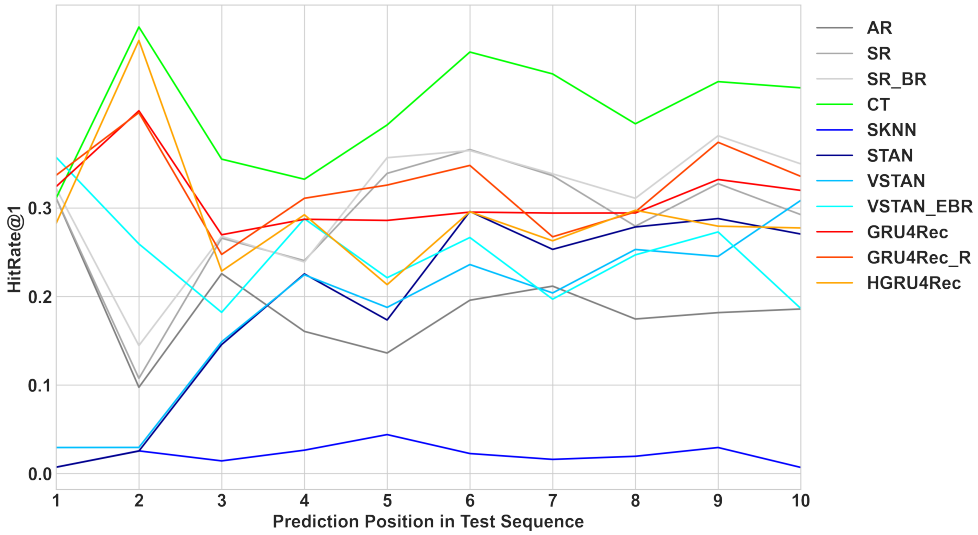

Figure 5: $HR@1$ performance across the first ten prediction positions on five-window app-level data.

and post-cutoff performance. Results in terms of $HR@1$ are displayed in Table 5. They are in line with individual-position results from Figure 5; however, we observe that performance for all neural models actually worsens for later positions in the test sequences. We conclude that neural network-based models struggle with the later positions in the prediction sequences. This is in line with our findings from above analyses, where we saw that neural methods also struggle with long prediction sequences - whenever training is not tailored towards them.

Algorithm	position ≤ 2	position > 2	position ≤ 5	position > 5	position ≤ 10	position > 10
AR	0.2269	0.1892	0.2082	0.1934	0.2045	0.1898
SR	0.2307	0.2768	0.2514	0.2798	0.2676	0.2516
SR_BR	0.2520	0.2854	0.2660	0.2904	0.2838	0.2468
CT	0.3878	0.3818	0.3766	0.4012	0.3911	0.3710
SKNN	0.0142	0.0167	0.0193	0.0111	0.0193	0.0061
STAN	0.0145	0.2298	0.0843	0.2602	0.1268	0.2385
VSTAN	0.0295	0.2230	0.0950	0.2469	0.1270	0.2577
VSTAN_EBR	0.3180	0.2058	0.2807	0.1903	0.2709	0.1405
GRU4Rec	0.3581	0.2984	0.3264	0.3098	0.3208	0.3173
GRU4Rec_R	0.3659	0.2827	0.3342	0.2816	0.3304	0.2311
HGRU4Rec	0.3639	0.2593	0.3132	0.2665	0.3073	0.2542

Table 5: $HR@1$ performance results on five-window app-level data, by positional cutoff within test sequence.

As a final piece of analysis regarding the positioning within test sequences, we consider the fact that around 38.91% of all app-level sequences (i.e., sessions) are ON-OFF sequences of length two, implying that a user only switches on their screen in order to switch it off immediately afterwards. This also means that 38.91% of all sequences have an OFF event in the second position of the original sequence or, equivalently, in the first prediction position of the sequence. Therefore, we expect that models learn this behavior and frequently predict an

OFF event as top-ranked recommendation in the first prediction position. We observe that this is indeed the case for most models and that models which do not learn this behavior are those with weak $HR@1$ performance. (The exact results are presented in Table 18 in the appendix.) It is crucial to note that the algorithms learn where to predict an OFF event and where not. In the sequence-level analysis in section 5.2 below we will see that the high prevalence of ON-OFF events and the fact that they can appear repeatedly within a sequence cause algorithms to simply predict ON-OFF events (almost) always; this is also one of the reasons why we conduct most sequence-level analyses with ON-OFF events removed, while we do not exclude ON and OFF events for the app-level analyses in Sections 5.1.2 and 5.1.3.

Removing ON and OFF Events

We have seen that short sequence length is a key issue and potential performance bottleneck for app-level analysis. Moreover, the first and last event of each sequence - ON and OFF events, respectively, by construction - are hardly informative and ON-OFF sequences are additionally very frequent.

To increase average sequence length as well as information density across and within sequences, here we analyze the effect of dropping all ON and OFF events from the app-level data. This automatically implies removing ON-OFF sequences as well as sequences of length three, because after removing the ON and OFF event such sequences only contain a single event and thus do not meet the required minimum sequence length of two anymore.

Figure 6 visualizes the effect of removing all ON and OFF events from training and evaluation data on $HR@k$ performance for $k = 1, 5, 10,$ and 20; Table 6 presents all performance metrics. We observe that for almost all models and metrics, dropping ON and OFF events increases performance, especially for the top-ranked recommendation (see $HR@1$). The increase in performance is particularly substantial for the nearest neighbor-based models *SKNN* and *STAN*. The explanation is as in the positional analysis above: short sessions, with the obligatory ON token as initial event, are hardly informative and thus make it difficult to effectively find similar sequences. Dropping many of these short sequences consequently increases performance. Finally, we notice that for almost all models there is a (sometimes substantial) increase in app coverage, with popularity bias decreasing strongly for all models; this is a consequence of removing the "popular" and thus frequently predicted OFF token.

While the removal of ON and OFF events leads to improvements in terms of model performance, it comes at a cost: model performance now reflects the models' capability to predict behavioral sequences *conditional* upon the sequence not being an ON-OFF sequence and upon the next event being neither an ON nor an OFF event. In our case, dropping ON and OFF events leads to the exclusion of 49.92% of all app-level sequences, implying that this conditional analysis is hardly representative of overall user behavior anymore. Therefore, and recalling the conclusion from the analysis of positioning within the test sequence, we decide against excluding

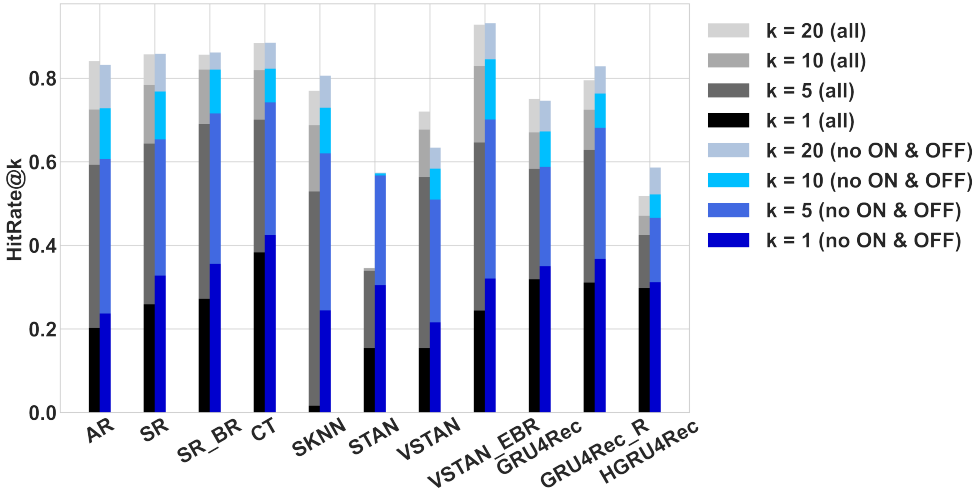

Figure 6: $HR@k$ performance comparison between full five-window app-level data (left bars) and five-window app-level data after dropping all ON and OFF events (right bars), for $k \in \{1, 5, 10, 20\}$.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.2367	0.6064	0.7282	0.8315	0.3834	0.3997	0.4071	0.4224	0.1373
SR	0.3274	0.6538	0.7680	0.8583	0.4485	0.4642	0.4705	0.4482	0.1301
SR_BR	0.3555	0.7156	0.8203	0.8613	0.4904	0.5050	0.5081	0.4581	0.1309
CT	**0.4244**	**0.7419**	0.8223	0.8844	**0.5502**	**0.5614**	**0.5656**	0.3711	0.1557
SKNN	0.2443	0.6201	0.7291	0.8057	0.3896	0.4042	0.4095	0.3070	0.1423
STAN	0.3049	0.5670	0.5721	0.5739	0.4147	0.4154	0.4156	0.1900	0.0523
VSTAN	0.2156	0.5093	0.5836	0.6338	0.3310	0.3411	0.3446	0.5333	0.0515
VSTAN_EBR	0.3203	0.7013	**0.8451**	**0.9316**	0.4628	0.4820	0.4881	0.6380	0.1020
GRU4Rec	0.3500	0.5875	0.6722	0.7460	0.4410	0.4524	0.4575	0.8776	0.0514
GRU4Rec_R	0.3673	0.6811	0.7630	0.8283	0.4889	0.5001	0.5047	0.9279	0.0766
HGRU4Rec	0.3118	0.4656	0.5220	0.5863	0.3712	0.3788	0.3831	0.6391	0.0218

Table 6: Performance results on five-window app-level data when dropping all ON and OFF events from both training and evaluation data. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

ON and OFF events for all subsequent app-level analyses.[13]

5.1.2 Category-level Prediction

Given that our ultimate goal is to predict human behavioral sequences, i.e., what type of behavior we can expect next given a particular phone usage pattern in the past, we might alternatively consider next-category prediction instead of next-app prediction. Using the app-level data representation, we are able to do so easily because for each event (app) we also have its app category at our disposal. This way, we do not predict a particular app anymore but

[13]For this comparison to be completely fair, one should additionally re-tune all models on the reduced dataset with ON and OFF events dropped. For runtime reasons, however, we decided against that. Furthermore, the key conclusion of this piece of analysis, i.e., increased performance at the cost of a loss of representativeness of user behavior, would not change.

simply the category - for instance, "Messaging" instead of "WhatsApp". If our algorithms perform significantly better on the category level, this would indicate that they actually learn more about behavioral sequences than what we concluded from the results in Section 5.1.1. The difference in performance would then be the extent to which the algorithms predict the right app category, yet get the specific app wrong. Furthermore, we would expect performance increases by non-session-aware models to be larger than those for session-aware models. This is because the long-term memory of session-aware models should, in theory, already learn specific app predilections within an app category on a user level (e.g., that user X mostly uses WhatsApp for messaging, whereas user Y mostly uses Telegram); since session-based models do not have this user-level memory, they should be more prone to getting the individual app wrong despite predicting the right category.

Note that by construction, category-level results cannot be worse than app-level results since whenever the correct app is predicted, the correct app category is automatically predicted as well - but not vice versa. Table 7 presents prediction results if we only consider the app category when calculating model performance.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20
AR	0.2224	0.6975	0.8150	0.9040	0.3890	0.4052	0.4113
SR	0.2822	0.7536	0.8471	0.9074	0.4623	0.4748	0.4791
SR_BR	0.2835	0.7599	**0.8660**	0.9081	0.4577	0.4728	0.4759
CT	**0.3984**	**0.7710**	0.8620	0.9216	**0.5464**	**0.5588**	**0.5630**
SKNN	0.0159	0.6319	0.7834	0.8503	0.2120	0.2327	0.2372
STAN	0.1581	0.3705	0.4174	0.4612	0.2440	0.2498	0.2530
VSTAN	0.1692	0.6066	0.7465	0.8013	0.3303	0.3498	0.3538
VSTAN_EBR	0.2443	0.6610	0.8607	**0.9448**	0.3868	0.4148	0.4209
GRU4Rec	0.3428	0.6569	0.7528	0.8321	0.4663	0.4795	0.4851
GRU4Rec_R	0.3370	0.6691	0.7806	0.8550	0.4606	0.4756	0.4810
HGRU4Rec	0.3285	0.4935	0.5668	0.6451	0.3923	0.4020	0.4075

Table 7: Performance results on five-window app-level data when only considering the category of apps in the test sequence. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

Comparing these results to the app-level results in Table 3, we observe that the performance increases for non-neural methods are rather small when looking at the top-ranked recommendation but noticeable for large recommendation list lengths, as illustrated in Figure 7. We moreover observe that, compared to the performance increase when dropping ON and OFF events, performance increases due to category-level prediction are approximately proportional to standard app-level performance, implying that none of the non-neural algorithms get the category right many more times than the actual app, in particular when only considering the top-ranked recommendation. For the neural network-based models, the increases due to category-level evaluation are more pronounced, especially for larger k, yet neither are they overwhelmingly large. Altogether, from this piece of analysis we conclude that next-app prediction performance as done in Section 5.1.1 is largely indicative of how much the algorithms truly learn in terms of behavioral sequence modeling. This holds for both session-based and session-aware models

as we do not find any evidence of larger performance increases for models without user-level memory.

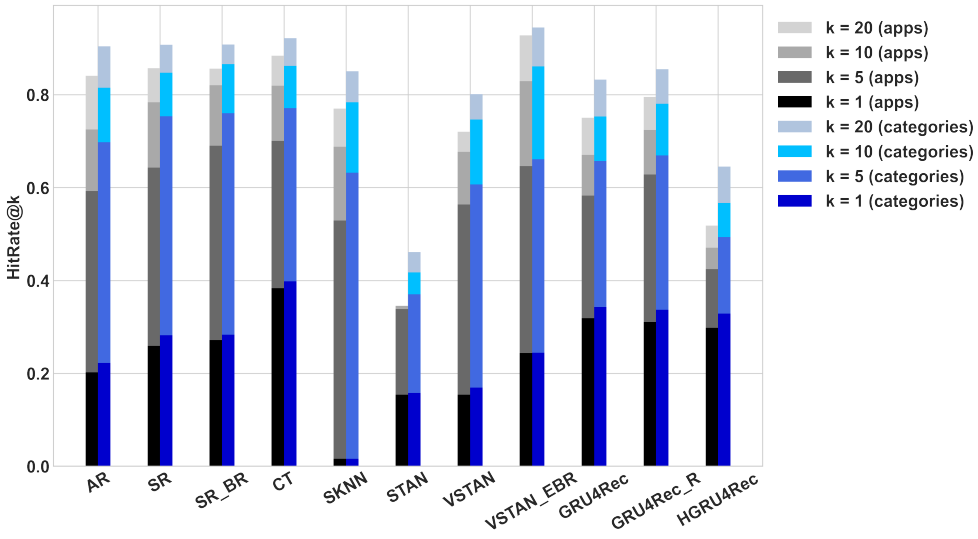

Figure 7: $HR@k$ performance increases on five-window app-level data when only considering app categories for evaluation (left bars) instead of considering the individual apps (right bars), for $k \in \{1, 5, 10, 20\}$.

5.1.3 Embedding Analysis of GRU4Rec

Having gained a general understanding of behavioral sequence modeling using next-app prediction, in this section we do not carry out any further prediction tasks. Instead, here we explore to which degree deep learning models learn smartphone app semantics in an illustrative fashion.

To do so, we look at *GRU4Rec* specifically and add an embedding layer of dimension 128 to the otherwise fully tuned model for training.[14] Here we use single-window data since we are interested in a single, exemplary embedding and thus averaging across five windows would not make much sense. We note that performance-wise, adding embeddings leads to negligible decreases.[15]

After extracting the embedding layer from the fully trained model, we apply the t-distributed stochastic neighbor embedding dimensionality reduction technique (TSNE; Hinton and Roweis, 2002) to obtain two-dimensional app embeddings. That is, for each app in the training set we now have a two-dimensional embedding vector, which lend themselves to intuitive visualization. In particular, we might want to know whether apps from a common app category - which

[14]The embedding layer was not part of the hyperparameter search space for *GRU4Rec*. Results for embedding dimensions of 16, 32, and 64 are similar and thus omitted.

[15]$HR@1$ performance, for instance, deteriorated by 0.64 percentage points compared to standard (non-embedding) single-window performance.

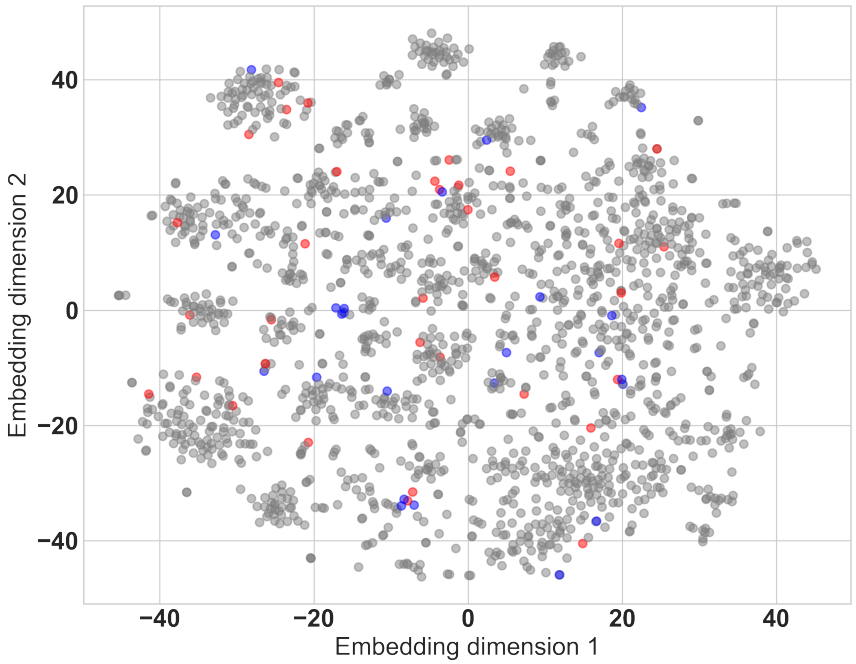

Figure 8: App category-based clustering of 128-dimensional app-level embeddings. Blue dots represent apps categorized as *Messaging*, red dots represent apps categorized as *Social Networks*. For illustration, app embeddings are reduced to a dimensionality of two, using TSNE.

by definition encompasses similar apps - form clusters in the embedding space.[16] Figure 8 exemplarily illustrates two-dimensional app embeddings for apps from the categories *Messaging* and *Social Networks*. No apparent category-level clustering is recognizable. In order to quantify this, we also consider the share of apps whose category coincides with the category of its most similar app, with similarity computed in terms of cosine similarity based on the original 128-dimensional embeddings. We find this share to be 11.67%, implying that only for less than one out of eight apps its nearest neighbor app is from the same category. As apps from the same category are, by construction, content-wise related (e.g., WhatsApp and Telegram), this implies that app embeddings learned by the *GRU4Rec* model are not very successful at learning true app-level semantics.

Alternatively, we might want to start off with a clustering approach which does not use pre-established app categories but rather clusters apps according to their positioning within the embedding space. We thus use k-means clustering on the untransformed 128-dimensional

[16]The need for individual events to be equipped with pre-established categories is the reason why we perform this embedding analysis only for app-level data.

app embeddings and cluster apps into 15 categories. An illustration of these clusters, using two-dimensional data points just as above, is provided in Figure 9.

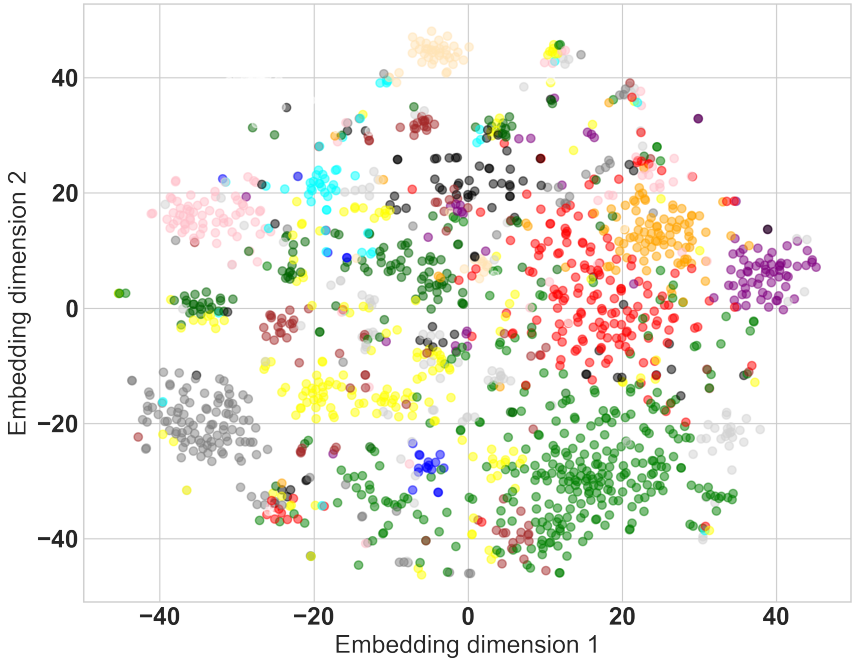

Figure 9: k-means clustering of app-level embeddings ($k = 15$). For illustration, app embeddings are reduced to a dimensionality of two, using TSNE.

To qualitatively assess whether these data-based clusters coincide with our pre-established app categories, we look at potential accumulations of app categories within each one of the 15 clusters. The small moccasin-colored cluster at the center top indeed demonstrates a high prevalence of apps from only few but related categories: out of a total 52 apps belonging to this cluster, 32 (i.e., more than 60%) are either camera or image editing apps. However, this logical coherence is much more the exception than the rule and, as can already be seen from the figure, most clusters are very much dispersed across the app space, with (almost) no intra-cluster app category clustering.

Finally, we briefly (re-)visit the concept of app analogies, by experimentally constructing app analogies such as "Messaging 1 - Social Network 1 + Social Networks 2 = ???". We find no meaningful app analogies in our embeddings, which does not come as a surprise: first, as mentioned in Section 2, app analogies are conceptually much less intuitive than word embeddings - see, for example, Table V of Ma et al. (2016). Second, the overall quality of our *GRU4Rec* embeddings is rather low, as evidenced by the lack of category-level clustering amongst apps.

Summing up our app-level analysis, we conclude that a variety of (modified) next-app pre-

diction tasks is possible and informative given our app-level data, implying that the algorithms do learn behavioral patterns based on app usage data. However, this learning - especially by neural network-based methods - is not profound enough as for app semantics to be recognizable.

5.2 Sequence-level Results

For sequence-level analysis, we again start by presenting overall results on the next-event prediction task. For reasons explained below, we address the effect of removing ON-OFF events immediately after that and conclude the section with an analysis of the impact of test sequence positioning. For sequence-level data, it is not necessary anymore to introduce and analyze a minimum sequence length. This is because the app-level to sequence-level data transformation we conducted already achieved a significant increase in sequence length. In fact, average sequence length is now 76.82 and only 14.74% of all sequences contain fewer than 20 events (and would thus be excluded by a minimum sequence length requirement of 20).

In the sequence-level setting, predicting the next event does not equate to next-app prediction anymore but instead to predicting the next (categorized) session of apps; that is, we now aim to model a user's subsequent session of smartphone activities. While the interpretation and the underlying data are clearly very different from the next-app prediction task, the technical analysis conducted - choice of algorithms, evaluation protocol, and metrics - is completely identical.

Overall Performance

Table 8 presents all five-window performance metrics of the tuned algorithms and Figure 10 shows $HR@k$ performance for different values of k.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.5295	0.6875	0.7541	0.8126	0.5891	0.5980	0.6021	0.4658	0.0733
SR	0.5305	0.7042	0.7653	0.8187	0.5981	0.6063	0.6101	0.8772	0.0724
SR_BR	**0.5307**	0.6669	0.7271	0.7931	0.5830	0.5911	0.5956	0.6703	0.0625
CT	**0.5307**	**0.7161**	**0.7705**	**0.8238**	**0.6040**	**0.6113**	**0.6150**	0.9467	0.0730
SKNN	0.5290	0.6917	0.7564	0.8149	0.5884	0.5970	0.6011	0.1807	0.0727
STAN	0.5277	0.6888	0.7391	0.7685	0.5879	0.5948	0.5969	0.5901	0.0666
VSTAN	0.5274	0.6831	0.7284	0.7415	0.5844	0.5907	0.5917	0.3240	0.0578
VSTAN_EBR	0.5301	0.6686	0.7225	0.7864	0.5830	0.5901	0.5946	0.6563	0.0620
GRU4Rec	0.5291	0.5298	0.5306	0.5317	0.5293	0.5294	0.5295	0.0818	0.0502
GRU4Rec_R	0.4982	0.6738	0.7381	0.7996	0.5666	0.5753	0.5796	0.5622	0.0661
HGRU4Rec	0.5134	0.5291	0.5360	0.5463	0.5194	0.5204	0.5211	0.9164	0.0510

Table 8: Performance results on five-window sequence-level data. Performance is averaged across five windows of equal time span. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

We observe a very strong performance in terms of $HR@1$, at around 50% for all algorithms; in fact, none of the algorithms in the app-level setting achieved such a high $HR@1$. Furthermore, we observe that performance increases with increasing k are much lower than in the app-level setting, leading to $HR@20$ performance being worse than in the app-level case for seven

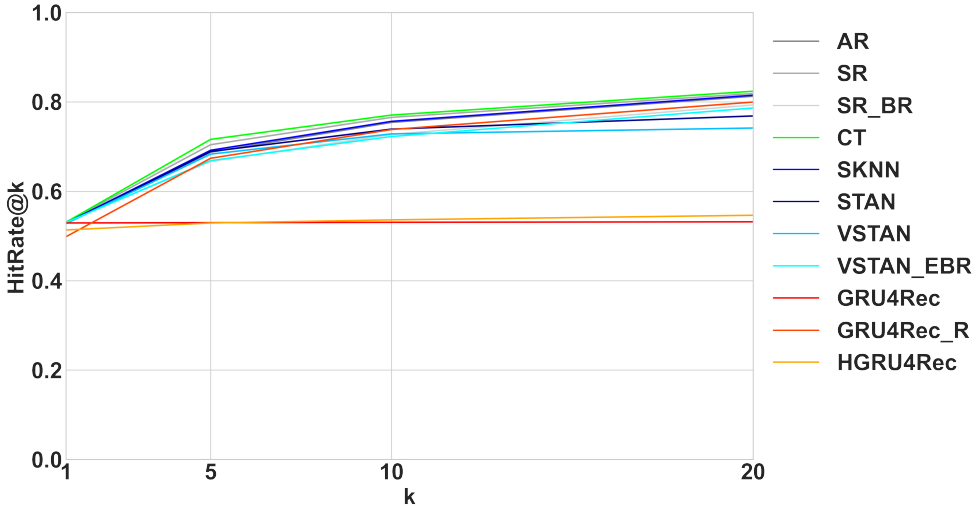

Figure 10: $HR@k$ performance for $k = 1, 5, 10,$ and 20 on five-window sequence-level data for all selected algorithms.

algorithms. The strongest performer across all metrics is the *CT* algorithm, just like in Section 5.1.1. *GRU4Rec* and *HGRU4Rec* are the weakest performers for all metrics where $k \geq 5$. This poor performance is surprising since sequence-level sequences are mostly rather long, facilitating - in principle - the learning for RNN-based models. Moreover, we observe that the *Remind* extension boosts *GRU4Rec* performance considerably whenever the recommendation list length is $k \geq 5$. In terms of the variety of tokens included in the recommendations, there are stark differences between the algorithms, with *CT* covering 95% of all tokens while *GRU4Rec* only covers 8%. Popularity bias is rather low (between 5% and 8%) for all models. Single-window results, as shown in Table 19 in the appendix, are very much in line with five-window results, except for the notable exception of *HGRU4Rec*, which performs considerably worse on the single-window data. (This, again, underlines the importance of the multiple-window approach.)

The high $HR@1$ performance across all algorithms, however, is suspicious because it might simply be the consequence of the high prevalence of ON-OFF tokens - instead of our models truly learning anything about behavioral sequences. For this reason, we analyze the effect of removing ON-OFF tokens immediately, that is, before looking at performance by position.

Removing ON-OFF Tokens

For sequence-level analysis, we do not have individual ON and OFF events anymore but rather ON-OFF tokens (i.e., events which represent a two-app, ON-OFF app-level session). Removing all ON and OFF events from the app-level data thus translates into removing all those ON-

OFF tokens from the sequence-level data.[17] Since these ON-OFF tokens constitute 51.06% of all tokens, the results as shown in Table 9 represent less than half of all observed sequence-level sequences (i.e., sentences) and are thus not representative of all behavioral sequences anymore. That said, the results - in particular when compared to those from Table 8 - are still very insightful: for all algorithms, performance drops significantly after excluding ON-OFF tokens.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.1993	0.4685	0.5804	0.6734	0.2995	0.3145	0.3209	0.4906	0.2551
SR	0.2496	**0.5017**	**0.6013**	**0.6874**	0.3442	0.3578	0.3638	0.8554	0.2437
SR_BR	0.2367	0.4914	0.5785	0.6664	0.3342	0.3460	0.3520	0.7060	0.1845
CT	**0.2539**	0.5006	0.5961	0.6838	**0.3462**	**0.3591**	**0.3652**	0.9128	0.2527
SKNN	0.1638	0.4348	0.5543	0.6541	0.2628	0.2789	0.2859	0.2571	0.2552
STAN	0.2148	0.4749	0.5833	0.6565	0.3098	0.3245	0.3297	0.8340	0.2175
VSTAN	0.1985	0.4391	0.5061	0.5311	0.2876	0.2970	0.2988	0.5103	0.1165
VSTAN_EBR	0.1941	0.4420	0.5331	0.6295	0.2858	0.2982	0.3048	0.6804	0.1699
GRU4Rec	0.1370	0.1900	0.1956	0.2024	0.1603	0.1610	0.1615	0.3598	0.0877
GRU4Rec_R	0.1843	0.4258	0.5273	0.6366	0.2738	0.2874	0.2950	0.6627	0.1649
HGRU4Rec	0.0586	0.1058	0.1264	0.1515	0.0765	0.0792	0.0809	0.9713	0.0563

Table 9: Performance results on five-window sequence-level data when dropping all ON and OFF events from the underlying app-level sessions. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

As illustrated in Figure 11, this drop is especially pronounced for $HR@1$ performance, with the average performance drop (across all algorithms) equalling 63.88%. This means that by virtue of removing all ON-OFF tokens, the number of behavioral sequences an average algorithm now correctly predicts drops by almost two-thirds, as compared to the previous scenario where no tokens are excluded. Across all algorithms and evaluation metrics, the performance obtained is now much worse than in the app-level setting. *CT* is still the best performer, while the neural methods *GRU4Rec* and *HGRU4Rec* are the worst performers for all metrics. One reason for the substantial performance drop of the neural network-based models is the fact that the ranking loss minimized during training does not take token frequency into account; and while this is not an issue for datasets with a large number of approximately equally likely tokens - e.g., standard RS datasets as in Hidasi et al. (2015) - it does become problematic for data such as ours, where some of the tokens are disproportionately frequent. An (inverse frequency-based) weighting of the loss contributions might thus help to alleviate this over-dependence on frequent tokens.

Judging by these results, it seems clear that the disproportional prevalence of ON-OFF tokens in the unrestricted data airbrushes the predictive power of the models while hampering true learning because models simply "learn" to predict an ON-OFF token every single time. In fact, it turns out that on the unrestricted sequence-level data, all algorithms predict an ON-OFF token as top ranked recommendation in more than 95% of all cases, except for *Gru4Rec_R* with a 89.27% ON-OFF prediction rate; the *GRU4Rec* algorithm even predicts ON-OFF events in 100% of all cases as its top recommendation. This in turn implies that the algorithms almost

[17]Since some tokens include multiple ON events, a secondary effect of stripping ON and OFF events from the underlying app-level sessions (which are then converted into tokens) is the reduction of the number of unique tokens. However, this effect is rather small, as only very few app-level sequences contain multiple ON events.

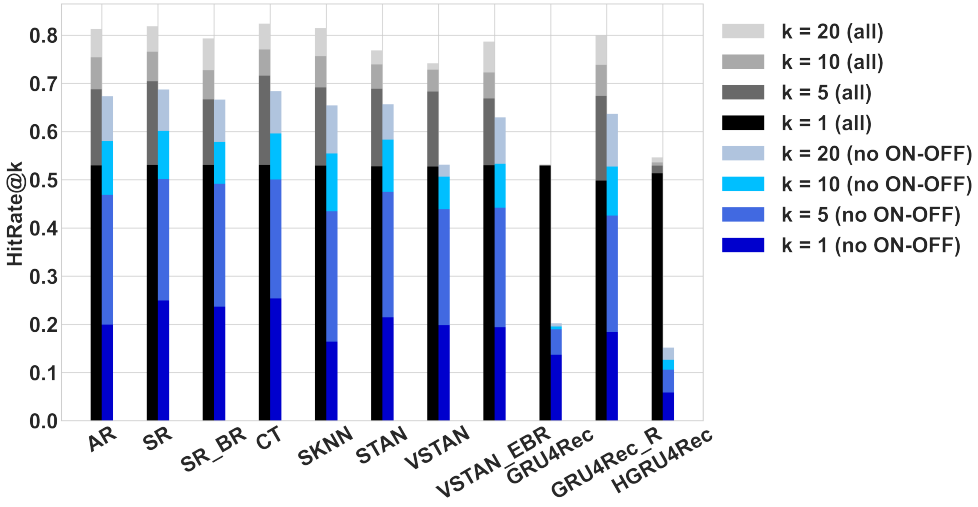

Figure 11: $HR@k$ performance comparison between full five-window sequence-level data (left bars) and five-window sequence-level data after dropping all ON and OFF events (right bars), for $k \in \{1, 5, 10, 20\}$.

never correctly predict non-ON-OFF events and, as a consequence, $HR@1$ on non-ON-OFF events is close to zero.

Despite the limited representativeness, we thus decide to use sequence-level data with all ON and OFF events removed from the underlying app-level data for the analysis of the effect of test sequence positions.

Position in Test Sequence

Like in the app-level setting, here we consider test sequence positions and how they affect model performance. The $HR@1$ results for the first ten prediction positions are shown in Figure 12 (see Table 20 in the appendix for the underlying numbers). As opposed to the app-level setting, however, we do not see a clear trend for any of the models as the position of the event within the test sequence increases. We do observe, however, that performance for the first position is relatively weak for all algorithms.

Next, we again look at positional cutoffs instead of individual positions. From the results in Table 10, again in terms of $HR@1$, we can now see that all models except *SKNN* perform better on later positions of the test sequences, with the precise positioning of the cutoff - after the second, fifth, or tenth position - not appearing to be influential. For the neural network-based models, this performance improvement for later events is in line with what we expected, yet it is precisely the opposite of what we observed for these models in the app-level setting. One potential explanation for the difference in performance across positions for app- versus sequence-level data is that in the app-level setting tuning and training was performed on predominantly short sequences, while in the sequence-level setting the large majority of sequences is long by

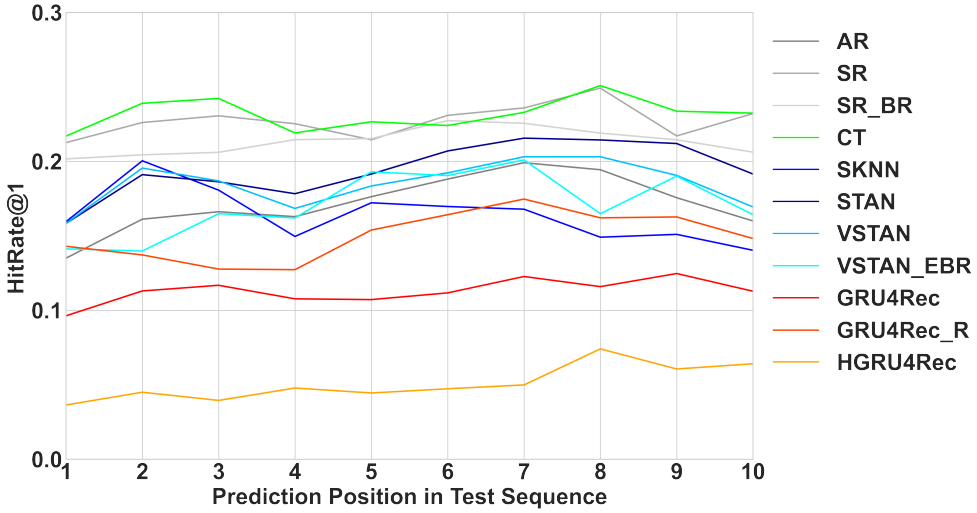

Figure 12: HR@1 performance across the first ten prediction positions on five-window sequence-level data for all selected algorithms.

construction. This would corroborate our conclusion from Section 5.1.1 that differences in sequence lengths between training and evaluation data negatively affect the performance of neural network-based algorithms.

Algorithm	position ≤ 2	position > 2	position ≤ 5	position > 5	position ≤ 10	position > 10
AR	0.4770	0.5310	0.4839	0.5328	0.4933	0.5351
SR	0.4760	0.5321	0.4833	0.5340	0.4944	0.5361
SR_BR	0.4767	0.5323	0.4856	0.5340	0.4964	0.5360
CT	0.4760	0.5323	0.4857	0.5340	0.4952	0.5362
SKNN	0.4509	0.5312	0.4743	0.5330	0.4900	0.5350
STAN	0.3819	0.5321	0.4351	0.5348	0.4726	0.5364
VSTAN	0.3823	0.5318	0.4345	0.5346	0.4718	0.5363
VSTAN_EBR	0.4776	0.5316	0.4861	0.5334	0.4952	0.5355
GRU4Rec	0.4777	0.5306	0.4838	0.5324	0.4931	0.5347
GRU4Rec_R	0.4506	0.4998	0.4574	0.5015	0.4712	0.5029
HGRU4Rec	0.3840	0.5172	0.4257	0.5200	0.4519	0.5231

Table 10: Performance results on five-window sequence-level data, by positional cutoff within test sequence.

However, the strong dependence of RNN-based model performance on training sequence length and the weak overall performance on sequence-level data might also hint towards an even bigger problem: the amount of data at our disposal might simply be insufficient for proper learning of behavioral sequences, in particular in the sequence-level setting where the dataset is much smaller again. In order to explore the full potential of the algorithms presented here and to examine the impact of dataset size, a larger study in terms of both number of users and

observation period is therefore needed.[18]

6 Discussion

6.1 Conclusion

In this paper, we explore how to model behavioral sequences through next-event prediction by using the PhoneStudy project data (Stachl et al., 2019), which contains smartphone app data for 310 participants over a 86-day period. Due to the sequentiality of the smartphone sessions - which impose a natural order on the data - and considering that there are almost 2,500 distinct events, we employ a broad selection of algorithms from the RS domain for modeling and next-event prediction.

We observe strong predictive performance of most algorithms across a variety of tasks, with $HR@1$ exceeding 40% and $HR@20$ exceeding 90% in some cases. Across all pieces of analysis, the *CT* algorithm performs particularly well, especially in terms of top-1 and top-5 recommendations. The co-occurrence frequency-based models *AR*, *SR*, and *SR_BR* also show strong overall performance, in particular when the recommendation list length is larger. The nearest neighbor-based *SKNN* algorithm is oftentimes the weakest of all models, with all other neighborhood-based models - being extensions of *SKNN* - performing better.

Regarding neural network-based models, we observe the following: first, they usually perform well in terms of $HR@1$ and $HR@5$, yet they never achieve top performance, always being outperformed by *CT*. *HGRU4Rec* is the worst-performing neural model most of the times, which is surprising since this session-aware method is supposed to be designed for this type of data with user-level information being available. One possible explanation is that *HGRU4Rec* does not yet employ the improved *TOP1-max* loss function used by *GRU4Rec*. Second, neural model performance is very sensitive to sequence length: in general, performance improves when sequence length of both training and evaluation sequences increases, yet it worsens if only the length of evaluation sequences increases but training is conducted on predominantly short sequences. Third, neural network-based models are amongst the worst performers in the sequence-level setting after dropping ON-OFF events, which might well be due to the small data size, with neural networks known to require large amounts of training data. And fourth, in terms of runtime and computational effort the neural algorithms are orders of magnitude more expensive than all other algorithms.

This leads us to conclude that non-neural models are the preferable modeling choice in our case, in line with the conclusions reached by Latifi et al. (2021) for their datasets. In particular, *CT* is recommendable for its strong performance when the desired recommendation list length

[18] For instance, the most recent PhoneStudy project, conducted by the Ludwig-Maximilians-University Munich and the Leibniz Institute for Psychology, collects smartphone usage data similar to the one used here, yet the observation period amounts to six months and the number of users is around 800.

is small (i.e., when $HR@1$ and $HR@5$ are of special interest) and because it requires no tuning. The co-occurrence frequency-based models are particularly suited because of their fast run-time and strong performance in terms of $HR@10$ and $HR@20$.

Finally, from the fact that session-aware models do not consistently outperform session-based models we conclude that there are no overarching user-level effects in our data. This in turn means that for predicting future behavioral sequences of a particular user, it is not overly helpful to know this particular person's past smartphone usage patterns. Still, the inexpensive heuristic user-level extensions we included are mostly effective, especially for short sequences and early positions. However, this is not due to these extensions adding some profound user-level learning to the session-based algorithms; instead, the extensions simply address some of the conceptual weaknesses. For instance, *VSTAN_EBR* alleviates the poor early-position performance of other neighborhood-based models, stemming from the low informational content in short sequences, through its extensions.

6.2 Limitations

While the overall objective - the explorative modeling of behavioral sequences - can be achieved given our algorithm selection and data, there are also limitations to this work.

First and foremost, the dataset size might simply be insufficient, also taking into account the low informational content of ON and OFF events. In particular for sequence-level analysis, this might be giving a relative advantage to non-neural methods, while neural methods might potentially benefit more than others from being trained on a larger dataset.

Second, our selection of algorithms far from covers the entire range of applicable RS methods. Especially in terms of deep learning-based RS, there are plenty of modern, usually sophisticated approaches we did not consider here, such as the attention-based *SHAN* (Xiao et al., 2019) and *BERT4Rec* (Sun et al., 2019). However, neural network-based models in general need plenty of training data, and attention-based models even more so. Furthermore, the main advantage of the attention mechanism over RNNs is its capability of handling long-term dependencies - which is precisely the opposite of the issue at hand, with our data predominantly containing very short app sessions. Because of that, we do not consider attention-based RS models as naturally applicable to our data and task.

Third, although we believe that our definition of an app session - beginning with an ON event and ending with an OFF event - is very reasonable, there might be other, more efficient ways to define and represent the raw data. For instance, one could try to take into account the duration and exact daytime of each app usage as well as to avoid the (almost) non-informative ON and OFF tokens.

6.3 Suggestions for Future Research

Based on the results from our analyses, there are two key points to be considered in future research: the size of the data and the amount of information an algorithm can incorporate.

As for the dataset size, at the end of Section 5.2 we already pointed to the most recent PhoneStudy project. Future research might want to use such sizeable datasets to corroborate the findings from our explorative analysis and, additionally, investigate the impact of data size on (neural network-based) model performance.

However, not only the mere size of a dataset matters, but also how much information can be extracted from it. Our data representation approach only uses an app's timestamp for the chronological ordering of a session. The exact timestamp, though, might be very informative about user behavior itself - be it the duration spent on an app or the exact moment of the day (or week) at which the app is opened. Moreover, additional information such as a user's geolocation might be incorporated. Unfortunately, the RS methods presented in this work are not capable of processing information beyond time-ordered event IDs. Therefore, converting all of the abovementioned information into natural language and subsequently feeding it to a (pre-trained) transformer, as suggested in Section 3.2.3, is certainly a promising direction for future research.

Altogether, when modeling behavioral sequences one can resort to a rich range of non-complex and strongly performing RS methods, while at the same time the topic remains a field of ample opportunities for new methods and research.

7 Appendix

Algorithm	Hyperparameter	App-level	Sequence-level
SR	steps	2	30
	weighting	div	div
SR_BR	steps	13	15
	weighting	quadratic	div
	boost_own_sessions	3.9	0.9
	reminders	True	True
	remind_strategy	hybrid	hybrid
	remind_sessions_num	1	10
	weight_IRec	0	1
SKNN	k	1500	100
	sample_size	500	500
STAN	k	100	100
	sample_size	10000	1000
	lambda_spw	7.24	0.4525
	lambda_snh	100	40
	lambda_inh	1e-05	0.4525
VSTAN	k	200	500
	sample_size	10000	10000
	similarity	vec	cosine
	lambda_spw	7.24	0.905
	lambda_snh	5	2.5
	lambda_inh	1.81	1e-05
	lambda_ipw	7.24	3.62
	lambda_idf	5	False
VSTAN_EBR	k	200	100
	sample_size	5000	2500
	similarity	vec	cosine
	lambda_spw	7.24	1.81
	lambda_snh	10	10
	lambda_inh	1.81	1e-05
	lambda_ipw	0.4525	3.62
	lambda_idf	2	False
	extend_session_length	24	6
	boost_own_sessions	2.5	3.1
	reminders	True	True
	remind_strategy	hybrid	hybrid
	weight_IRec	2	3
	weight_SSim	3	3

Table 11: Optimal hyperparameters for non-neural models for app- and sequence-level data.

Algorithm	Hyperparameter	App-level	Sequence-level
GRU4Rec	loss	bpr-max	top1-max
	final_act	elu-0.5	linear
	dropout_p_hidden	0.4	0.7
	momentum	0.0	0.0
	learning_rate	0.01	0.4
	constrained_embedding	False	True
GRU4Rec_R	loss	bpr-max	top1-max
	final_act	elu-0.5	elu-0.5
	dropout_p_hidden	0.1	0.7
	momentum	0.4	0.2
	learning_rate	0.01	0.01
	constrained_embedding	True	True
	reminders	True	True
	remind_strategy	hybrid	hybrid
	remind_sessions_num	9	4
	weight_IRec	1	3
HGRU4Rec	final_act	relu	linear
	dropout_p_hidden_usr	0.0	0.2
	dropout_p_hidden_ses	0.2	0.7
	dropout_p_init	0.3	0.1
	momentum	0.1	0.6
	learning_rate	0.03	0.5
	user_propagation_mode	init	all
	batch_size	100	50

Table 12: Optimal hyperparameters for neural network-based models for app- and sequence-level data.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.1928	0.5752	0.7256	0.8432	0.3227	0.3431	0.3514	0.1794	0.2318
SR	0.2400	0.6264	0.7944	0.8616	0.3953	0.4185	0.4234	0.2053	0.1879
SR_BR	0.2616	0.6848	**0.8312**	0.8600	0.4249	0.4457	0.4479	0.2076	0.1895
CT	**0.3608**	**0.6968**	0.8104	0.8848	**0.5045**	**0.5201**	**0.5255**	0.1990	0.2470
SKNN	0.0032	0.5320	0.6736	0.7608	0.1909	0.2112	0.2171	0.0201	0.2395
STAN	0.1400	0.2896	0.2920	0.2920	0.2025	0.2029	0.2029	0.0656	0.0876
VSTAN	0.1464	0.5560	0.6408	0.6800	0.3012	0.3130	0.3157	0.1420	0.1826
VSTAN_EBR	0.2440	0.6176	0.8056	**0.8936**	0.3777	0.4051	0.4115	0.3726	0.2123
GRU4Rec	0.2784	0.5208	0.6432	0.7320	0.3750	0.3923	0.3985	0.6504	0.0897
GRU4Rec_R	0.2464	0.5280	0.6184	0.6944	0.3539	0.3663	0.3718	0.6446	0.1007
HGRU4Rec	0.2992	0.4344	0.4728	0.5056	0.3520	0.3574	0.3596	0.5446	0.0220

Table 13: Performance results on single-window app-level data. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	-0.0096	-0.0174	0.0006	0.0025	-0.0147	-0.0124	-0.0123	-0.1414	-0.0010
SR	-0.0191	-0.0167	0.0108	0.0048	-0.0103	-0.0063	-0.0066	-0.1499	-0.0013
SR_BR	-0.0102	-0.0054	0.0109	0.0041	-0.0065	-0.0039	-0.0044	-0.1502	0.0003
CT	-0.0225	-0.0038	-0.0089	0.0008	-0.0085	-0.0089	-0.0081	-0.1277	-0.0021
SKNN	-0.0127	0.0030	-0.0140	-0.0089	0.0000	-0.0007	-0.0007	-0.0288	-0.0068
STAN	-0.0143	-0.0490	-0.0531	-0.0539	-0.0282	-0.0288	-0.0288	-0.0739	-0.0007
VSTAN	-0.0079	-0.0075	-0.0361	-0.0402	-0.0074	-0.0114	-0.0119	-0.2234	-0.0012
VSTAN_EBR	0.0004	-0.0286	-0.0234	-0.0342	-0.0045	-0.0026	-0.0035	-0.1850	-0.0007
GRU4Rec	-0.0406	-0.0621	-0.0269	-0.0183	-0.0468	-0.0415	-0.0409	-0.1498	0.0002
GRU4Rec_R	-0.0641	-0.1000	-0.1057	-0.1005	-0.0746	-0.0754	-0.0749	-0.1754	-0.0431
HGRU4Rec	0.0014	0.0097	0.0022	-0.0126	0.0034	0.0026	0.0016	-0.1378	-0.0029

Table 14: Differences between single- and multiple-window performance on app-level data.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.2265	0.5717	0.6721	0.7905	0.3648	0.3784	0.3868	0.6149	0.1840
SR	0.3140	0.6303	0.7390	0.8256	0.4337	0.4482	0.4544	0.6447	0.1618
SR_BR	0.3419	0.6804	0.7858	0.8280	0.4696	0.4841	0.4872	0.6688	0.1668
CT	**0.4257**	**0.7222**	**0.7925**	0.8523	**0.5443**	**0.5539**	**0.5580**	0.6521	0.2048
SKNN	0.0130	0.3480	0.5366	0.6849	0.1117	0.1379	0.1483	0.0609	0.2070
STAN	0.2570	0.6350	0.6595	0.6610	0.4085	0.4121	0.4122	0.3316	0.0707
VSTAN	0.0977	0.3368	0.3992	0.4438	0.1836	0.1922	0.1954	0.4884	0.0373
VSTAN_EBR	0.2232	0.5737	0.7242	**0.8788**	0.3466	0.3665	0.3775	0.7121	0.1295
GRU4Rec	0.3693	0.6093	0.6840	0.7616	0.4651	0.4752	0.4806	0.8887	0.0610
GRU4Rec_R	0.3516	0.6612	0.7673	0.8451	0.4673	0.4817	0.4873	0.9722	0.1050
HGRU4Rec	0.3067	0.4602	0.5073	0.5595	0.3666	0.3729	0.3765	0.7059	0.0224

Table 15: Performance results on five-window app-level data when only considering sequences of at least 20 events for both training and evaluation. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.1890	0.4647	0.5921	0.7895	0.2964	0.3135	0.3274	0.1315	0.2349
SR	0.1810	0.4747	0.6646	0.7987	0.2932	0.3176	0.3272	0.1665	0.1832
SR_BR	0.1903	0.5183	0.7357	0.7974	0.3158	0.3453	0.3498	0.1633	0.1833
CT	**0.3123**	**0.5908**	0.7314	0.8492	**0.4221**	**0.4405**	**0.4492**	0.1608	0.2460
SKNN	0.0042	0.2962	0.5733	0.6433	0.0791	0.1140	0.1190	0.0384	0.2462
STAN	0.2440	0.6113	0.6330	0.6330	0.3908	0.3940	0.3940	0.0738	0.0959
VSTAN	0.2307	0.6056	**0.7499**	0.8178	0.3803	0.3999	0.4048	0.2012	0.2010
VSTAN_EBR	0.1345	0.4392	0.7024	**0.9016**	0.2385	0.2740	0.2886	0.1467	0.2141
GRU4Rec	0.2846	0.5013	0.5874	0.6798	0.3683	0.3797	0.3860	0.4380	0.0477
GRU4Rec_R	0.2091	0.4544	0.5679	0.6695	0.2963	0.3116	0.3186	0.4580	0.1213
HGRU4Rec	0.2716	0.4096	0.4696	0.5225	0.3251	0.3330	0.3366	0.4225	0.0165

Table 16: Performance on five-window app-level data when training on all sequences but only considering sequences of at least 20 events for evaluation. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

Algorithm	position = 1	position = 2	position = 3	position = 4	position = 5	position = 6	position = 7	position = 8	position = 9	position = 10
AR	0.3110	0.0974	0.2257	0.1605	0.1361	0.1957	0.2116	0.1745	0.1817	0.1858
SR	0.3110	0.1075	0.2657	0.2404	0.3389	0.3658	0.3363	0.2793	0.3273	0.2923
SR_BR	0.3218	0.1445	0.2678	0.2389	0.3566	0.3647	0.3384	0.3108	0.3814	0.3496
CT	0.3110	0.5042	0.3550	0.3323	0.3936	0.4759	0.4512	0.3949	0.4425	0.4355
SKNN	0.0072	0.0256	0.0143	0.0264	0.0440	0.0226	0.0160	0.0195	0.0294	0.0069
STAN	0.0072	0.0254	0.1456	0.2254	0.1734	0.2959	0.2531	0.2784	0.2879	0.2704
VSTAN	0.0294	0.0295	0.1488	0.2244	0.1876	0.2360	0.2039	0.2530	0.2452	0.3085
VSTAN_EBR	0.3571	0.2593	0.1820	0.2877	0.2211	0.2665	0.1969	0.2469	0.2728	0.1863
GRU4Rec	0.3240	0.4096	0.2696	0.2871	0.2858	0.2952	0.2941	0.2942	0.3320	0.3197
GRU4Rec_R	0.3368	0.4075	0.2474	0.3108	0.3257	0.3479	0.2673	0.2954	0.3740	0.3355
HGRU4Rec	0.2825	0.4890	0.2286	0.2922	0.2133	0.2956	0.2628	0.2973	0.2793	0.2773

Table 17: HR@1 performance results on five-window app-level data, by position within test sequence.

Algorithm	position = 1	position = 2	position = 3	position = 4	position = 5	position = 6	position = 7	position = 8	position = 9	position = 10
AR	0.8493	0.0016	0.0000	0.0017	0.0045	0.0082	0.0000	0.0000	0.0000	0.0000
SR	0.8493	0.3231	0.2359	0.3297	0.4489	0.3644	0.4825	0.3673	0.3897	0.4217
SR_BR	0.8558	0.3570	0.2599	0.3595	0.4693	0.4204	0.4966	0.3751	0.3978	0.4460
CT	0.8838	0.3170	0.2386	0.3189	0.4134	0.3261	0.4196	0.3391	0.3424	0.3539
SKNN	0.0000	0.0663	0.1248	0.1358	0.1333	0.1258	0.1360	0.1317	0.1235	0.1034
STAN	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
VSTAN	0.1507	0.0361	0.0067	0.0054	0.0069	0.0000	0.0000	0.0000	0.0000	0.0000
VSTAN_EBR	0.6067	0.4888	0.4201	0.3579	0.3092	0.2598	0.2694	0.2491	0.2170	0.2083
GRU4Rec	1.0000	0.0898	0.0548	0.0295	0.0698	0.0278	0.0000	0.0186	0.0105	0.0118
GRU4Rec_R	0.9377	0.0850	0.1550	0.2178	0.3411	0.2335	0.1923	0.1735	0.1405	0.1540
HGRU4Rec	0.0439	0.0479	0.0226	0.0085	0.0021	0.0032	0.0107	0.0000	0.0000	0.0000

Table 18: Relative frequency of predicting an OFF event as top-ranked recommendation on five-window app-level data, by position in the test sequence.

Algorithm	HitRate@1	HitRate@5	HitRate@10	HitRate@20	MRR@5	MRR@10	MRR@20	Coverage@20	Popularity@20
AR	0.5189	0.6781	0.7420	0.7988	0.5796	0.5882	0.5921	0.2237	0.0733
SR	0.5188	0.6933	0.7540	0.8058	0.5878	0.5959	0.5995	0.6626	0.0724
SR_BR	0.5202	0.6533	0.7055	0.7725	0.5713	0.5783	0.5829	0.5253	0.0618
CT	**0.5203**	**0.7083**	**0.7630**	**0.8124**	**0.5952**	**0.6026**	**0.6061**	0.8867	0.0725
SKNN	0.5188	0.6761	0.7388	0.7907	0.5741	0.5825	0.5861	0.0990	0.0722
STAN	0.5143	0.6613	0.7121	0.7348	0.5680	0.5750	0.5767	0.4185	0.0648
VSTAN	0.5143	0.6558	0.7025	0.7131	0.5666	0.5730	0.5739	0.1842	0.0567
VSTAN_EBR	0.5177	0.6465	0.6950	0.7601	0.5680	0.5744	0.5789	0.5428	0.0614
GRU4Rec	0.5183	0.5190	0.5193	0.5197	0.5186	0.5187	0.5187	0.0432	0.0502
GRU4Rec_R	0.4918	0.6603	0.7198	0.7820	0.5549	0.5628	0.5671	0.4723	0.0636
HGRU4Rec	0.3240	0.3424	0.3512	0.3611	0.3309	0.3321	0.3328	0.9703	0.0343

Table 19: Performance results on single-window sequence-level data. For $HR@k$ and $MRR@k$, the best performance per metric is highlighted in bold.

Algorithm	position = 1	position = 2	position = 3	position = 4	position = 5	position = 6	position = 7	position = 8	position = 9	position = 10
AR	0.1350	0.1612	0.1662	0.1629	0.1763	0.1881	0.1991	0.1944	0.1756	0.1600
SR	0.2126	0.2261	0.2306	0.2253	0.2145	0.2309	0.2359	0.2492	0.2171	0.2321
SR_BR	0.2017	0.2044	0.2061	0.2146	0.2153	0.2274	0.2256	0.2190	0.2145	0.2062
CT	0.2170	0.2390	0.2423	0.2191	0.2266	0.2241	0.2329	0.2509	0.2337	0.2324
SKNN	0.1595	0.2004	0.1807	0.1496	0.1722	0.1697	0.1679	0.1491	0.1510	0.1403
STAN	0.1583	0.1911	0.1863	0.1783	0.1915	0.2070	0.2156	0.2144	0.2120	0.1915
VSTAN	0.1583	0.1955	0.1869	0.1684	0.1835	0.1923	0.2031	0.2031	0.1906	0.1694
VSTAN_EBR	0.1413	0.1398	0.1649	0.1620	0.1930	0.1905	0.2010	0.1649	0.1901	0.1642
GRU4Rec	0.0963	0.1130	0.1168	0.1077	0.1072	0.1117	0.1227	0.1159	0.1247	0.1128
GRU4Rec_R	0.1430	0.1372	0.1277	0.1273	0.1539	0.1642	0.1747	0.1621	0.1627	0.1482
HGRU4Rec	0.0364	0.0450	0.0395	0.0478	0.0445	0.0473	0.0499	0.0741	0.0606	0.0641

Table 20: Performance results on five-window sequence-level data, by position within test sequence.

8 Declaration of Originality

I hereby declare that this thesis represents my own work and that I have documented all sources and materials used. I have clearly referenced in accordance with departmental requirements. Additionally, I confirm that this work has not previously been presented to another examination board and has not been published. I confirm that I understand that my work may be electronically checked for plagiarism and appreciate that any false claim regarding this work will result in disciplinary action.

Munich, September 28, 2021 Simon Wiegrebe

References

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. Predicting the next app that you are going to use. In *Proceedings of the eighth ACM international conference on web search and data mining*, pages 285–294, 2015.

Geoffray Bonnin and Dietmar Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):1–35, 2014.

Gokul Chittaranjan, Jan Blom, and Daniel Gatica-Perez. Mining large-scale smartphone data for personality studies. *Personal and Ubiquitous Computing*, 17(3):433–450, 2013.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Yves-Alexandre de Montjoye, Jordi Quoidbach, Florent Robic, and Alex Sandy Pentland. Predicting personality using novel mobile phone-based metrics. In *International conference on social computing, behavioral-cultural modeling, and prediction*, pages 48–55. Springer, 2013.

Christos Dimitrakakis. Bayesian variable order markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 161–168. JMLR Workshop and Conference Proceedings, 2010.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. Sequence and time aware neighborhood for session-based recommendations: Stan. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1069–1072, 2019.

Yupu Guo, Duolong Zhang, Yanxiang Ling, and Honghui Chen. A joint neural network for session-aware recommendation. *IEEE Access*, 8:74205–74215, 2020.

Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 843–852, 2018.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Geoffrey Hinton and Sam T Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer, 2002.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

Liang Hu, Qingkui Chen, Haiyan Zhao, Songlei Jian, Longbing Cao, and Jian Cao. Neural cross-session filtering: Next-item prediction under intra-and inter-session context. *IEEE Intelligent Systems*, 33(6):57–67, 2018.

Dietmar Jannach and Malte Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 306–310, 2017.

Dietmar Jannach, Malte Ludewig, and Lukas Lerche. Session-based item recommendation in e-commerce: on short-term intents, reminders, trends and discounts. *User Modeling and User-Adapted Interaction*, 27(3):351–392, 2017.

Iman Kamehkhosh, Dietmar Jannach, and Malte Ludewig. A comparison of frequent pattern techniques and a deep learning method for session-based recommendation. In *RecTemp@ RecSys*, pages 50–56, 2017.

Sara Latifi, Noemi Mauro, and Dietmar Jannach. Session-aware recommendation: A surprising quest for the state-of-the-art. *Information Sciences*, 573:291–315, 2021.

Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428, 2017.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021.

Malte Ludewig and Dietmar Jannach. Evaluation of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 28(4-5):331–390, 2018.

Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. Empirical analysis of session-based recommendation algorithms. *User Modeling and User-Adapted Interaction*, 31(1):149–181, 2021.

Qiang Ma, Shanmugavelayutham Muthukrishnan, and Wil Simpson. App2vec: Vector modeling of mobile apps and applications. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 599–606. IEEE, 2016.

Rischan Mafrur, I Gde Dharma Nugraha, and Deokjai Choi. Modeling and discovering human behavior from smartphone sensing life-log data for identification purpose. *Human-centric Computing and Information Sciences*, 5(1):1–18, 2015.

Fei Mi and Boi Faltings. Context tree for adaptive session-based recommendation. *arXiv preprint arXiv:1806.03733*, 2018.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Using sequential and non-sequential patterns in predictive web usage mining tasks. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 669–672. IEEE, 2002.

Christian Montag, Harald Baumeister, Christopher Kannen, Rayna Sariyska, Eva-Maria Meßner, and Matthias Brand. Concept, possibilities and pilot-testing of a new smartphone application for the social and life sciences to study human behavior including validation data from personality psychology. *J*, 2(2):102–115, 2019.

Nagarajan Natarajan, Donghyuk Shin, and Inderjit S Dhillon. Which app will you use next? collaborative filtering with interactional context. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 201–208, 2013.

Ella Peltonen, Parsa Sharmila, Kennedy Opoku Asare, Aku Visuri, Eemil Lagerspetz, and Denzil Ferreira. When phones get personal: Predicting big five personality traits from application usage. *Pervasive and Mobile Computing*, 69:101269, 2020.

Tu Minh Phuong, Tran Cong Thanh, and Ngo Xuan Bach. Combining user-based and session-based recommendations with recurrent neural networks. In *International Conference on Neural Information Processing*, pages 487–498. Springer, 2018.

Tu Minh Phuong, Tran Cong Thanh, and Ngo Xuan Bach. Neural session-aware recommendation. *IEEE Access*, 7:86884–86896, 2019.

Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137, 2017.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

Massimiliano Ruocco, Ole Steinar Lillestøl Skrede, and Helge Langseth. Inter-session modeling for session-based recommendation. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 24–31, 2017.

Guy Shani, David Heckerman, Ronen I Brafman, and Craig Boutilier. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(9), 2005.

Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *arXiv preprint arXiv:2106.03253*, 2021.

Clemens Stachl, Ramona Schoedel, Quay Au, Sarah Völkel, Daniel Buschek, Heinrich Hussmann, Bernd Bischl, and Markus Bühner. The phonestudy project, Aug 2019. URL `osf.io/ut42y`.

Clemens Stachl, Quay Au, Ramona Schoedel, Samuel D Gosling, Gabriella M Harari, Daniel Buschek, Sarah Theres Völkel, Tobias Schuwerk, Michelle Oldemeier, Theresa Ullmann, et al. Predicting personality from patterns of behavior collected with smartphones. *Proceedings of the National Academy of Sciences*, 117(30):17680–17687, 2020.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

Siliang Tong, Xueming Luo, and Bo Xu. Personalized mobile marketing strategies. *Journal of the Academy of Marketing Science*, 48(1):64–78, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Teng Xiao, Shangsong Liang, and Zaiqiao Meng. Hierarchical neural variational model for personalized sequential recommendation. In *The World Wide Web Conference*, pages 3377–3383, 2019.

Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple but hard-to-beat baseline for session-based recommendations. *arXiv preprint arXiv:1808.05163*, 2018.