PREDICTING THE DWELL TIME FOR THE NEXT MOBILE PHONE APPLICATION

USING LONG SHORT-TERM MEMORY NEURAL NETWORKS

KATHARINA PRITZL

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN DATA SCIENCE & SOCIETY
AT THE SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
OF TILBURG UNIVERSITY

STUDENT NUMBER

2077635

COMMITTEE

dr. Drew Hendrickson
Büşra Özgöde Yigin

LOCATION

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science &
Artificial Intelligence
Tilburg, The Netherlands

DATE

June 24, 2022

WORD COUNT

8761/8800

ACKNOWLEDGMENTS

I would like to sincerely thank my supervisor, Dr. Drew Hendrickson, for his guidance and support during this semester. His intellectually challenging ideas always helped me to stand up to all the challenges I encountered. I also want to give my thanks to George Aalbers for providing me the dataset which was used in this thesis. Finally, I would like to thank my partner, Konrad, and my mother, for their unconditional support, without which I would not have been able to finish my degree with the results I obtained.

CONTENTS

PREDICTING THE DWELL TIME FOR THE NEXT MOBILE PHONE APPLICATION

USING LONG SHORT-TERM MEMORY NEURAL NETWORKS

KATHARINA PRITZL

Abstract

Although prior research has addressed the task of next-app prediction, the area of predicting app engagement is widely unresearched. Since engagement predictions for mobile-phone applications can help to personalize features related to app usage, thus leading to a higher user satisfaction, the present work introduces a deep-learning approach to address this research gap. By proposing a generic LSTM that is capable of including multiple independent app usage sequences, the multiclass classification problem of dwell time prediction is addressed. Not only a solution for the cold-start problem, but also a framework for other areas where vast amounts of independent sequences have to be processed is introduced. Additionally, the trade-off between users' privacy, accuracy, and computational efficiency is investigated by comparing the performance of the proposed LSTM and an MLP trained on different feature subsets. As the basis for this research, a dataset of 186 users with over four million app records was used. While the LSTM outperformed the MLP in all tests, and was particularly suited for working on a reduced feature subset, the best performing LSTM reached an accuracy of 0.46. For the cold-start problem, an accuracy of 0.41 could be reached.

1 INTRODUCTION AND RESEARCH GOAL

Since the deployment of the first smartphones in the early 2000s, the use of mobile phone applications has become an indispensable part of everyday life. In reaction to the growing number of available mobile phone applications, research in the area of machine learning has focused on how engaging with smartphones can be made more comfortable by developing app recommendation systems based on the prediction task of determining which app a user is going to open next (Cao & Lin, 2017).
Research related to next-app predictions can mainly be utilized for an

improvement of smartphone operating systems. However, for app service providers, developing insights into how users engage with a specific application is more important than knowing which app a user is going to open next. One simple but strong metric for measuring users engagement with online services is the dwell time, the time a user spends engaging with a service (Lalmas, O'Brien, & Yom-Tov, 2014).
Although the area of next-app predictions is well researched, including the use of novel deep-learning approaches (Katsarou, Yu, & Beierle, 2022; Lee, Cho, & Choi, 2019; Xu et al., 2020), surprisingly little research has been conducted in the area of predicting app engagement. At this time, only two articles have addressed a related task by relying on traditional machine-learning models (Mathur, Lane, & Kawsar, 2016; Tian, Zhou, & Pelleg, 2021). A shortcoming of those is that they neither investigate the application of deep learning models, nor do they mention how to deal with the cold-start problem, a situation where predictions have to be made for users who were not included in the training data.
Accordingly, there is still a research gap related to app engagement predictions, a gap that becomes especially interesting by pointing to the recent prominence of LSTMs for tackling sequential prediction tasks. In response to that, the main research goal of this paper is to adapt a deep-learning approach for predicting the dwell time for smartphones on an application-based and user-independent level treated as a multiclass classification problem.

1.1 *Motivation and Relevance*

Getting in-depth insights into users' app engagement is, from a business perspective, crucial for the long-term success of mobile phone applications. Being able to forecast the time a user will stay on an app enables app service providers to better plan when to place advertisements, in-app notifications, or when to display which content. Personalizing those features can increase the users' satisfaction, leading to longer and more frequent usage, better reviews, and, therefore, to the long-time success of the app.
From a societal and scientific perspective, it is particularly important to propose an approach for improving users' satisfaction without sacrificing their privacy. Therefore, the present research will investigate which information is truly necessary for generating reliable predictions. Taking a critical look at the trade-off between computational efficiency, accuracy, and protection of sensitive data is a crucial part of practicing responsible AI, where it is the responsibility of the scientific community to fulfil the rising societal demands for privacy and explainability (Arrieta et al., 2020). In particular, when using deep-learning models, known to be black boxes,

a special awareness for the used features has to be raised. While this research is not focused on increasing the explainabilty of those models, it does try to propose an approach for reducing the amount of information needed paying respect to privacy concerns. To preserve computational resources where possible, it additionally investigates if a complex and computationally more demanding deep-learning structure trained on a sequential version of the data is indeed advantageous compared to a simpler one.
Thereby, the framework for a model capable of including multiple independent sequences can not only be used in the area of smartphone predictions but can also be adapted to other domains. Since including multiple data sequences in one model comes with a risk of biasing the results towards the longer ones, the aim of the present work is to develop a framework for preventing such a bias, which is particularly important from a scientific viewpoint.

1.2 *Research Questions*

Based on the outlined research gap and the demonstrated societal relevance, the main research question of the present work is as follows:

MQ *To what extent is it possible to predict the dwell time for the next mobile-phone application a user is going to open, based on a combination of historical app-log data, contextual, and user-related features with an LSTM trained by following a generic modeling approach?*

A commonly used distinction to investigate feature-relevance in the field of smartphone predictions is to differentiate between user-related, contextual, and app-history related features. While user-related features describe features related to characteristics of the smartphone user, the term contextual refers to features like time and location. App history-related features refers to characteristics related to previous app usage.

The research will be guided by the sub-questions shown below:

RQ1 *To what extent will the use of an LSTM architecture improve the accuracy for dwell time predictions on an application-based level compared to a feed-forward neural network, namely MLP, both trained user-independently?*

Guided by the research gap in the literature, the aim of the present work is to adapt the use of neural networks for the prediction task of app engagement. In particular, the performance of a more computationally demanding LSTM, which is capable of capturing long-term dependencies by utilizing the sequential character of the data, will be explored and

compared to the performance of an MLP that has no access to the sequential app history and is more computationally effective.

RQ2 *To what extent will the proposed user-independent LSTM structure bias the results towards users for whom more training data is available?*

To check whether the proposed generic modeling approach for the LSTM is biasing the results, an error analysis dedicated to this question will be conducted.

RQ3 *Which feature subset can reach the best predictive performance for the task of dwell time prediction for both MLP and LSTM, respectively, is it possible to minimize user-related and contextual features to prevent privacy concerns while keeping the accuracy of the proposed models high, and what differences can be seen between both models using different feature subsets?*

The trade-off between model complexity, accuracy, and user privacy is investigated in detail by training the proposed LSTM and an MLP on different feature subsets.

RQ4 *To what extent can the proposed LSTM structure solve the cold-start problem?*

Since training individual models for each user is both computationally inefficient and not suited for getting predictions for unseen users, the predictive performance of the proposed generic LSTM will be tested on the cold-start problem.
All abbreviations which are used throughout the present work can be found in Appendix A.

2 LITERATURE REVIEW

The area of predicting engagement for smartphone users is widely unresearched. To the best of my knowledge, only Tian et al. (2021) and Mathur et al. (2016) addressed similar tasks. Accordingly, the following section will first explore their works before switching to a broader scope by discussing two related and highly researched areas: next-app predictions and dwell time predictions for streaming services. While the findings of works related to the area of next-app prediction provide insightful domain knowledge needed for an in-depth understanding of smartphone usage, and how it can be modeled for deep-learning approaches, the field of dwell time prediction helps to illuminate challenges related to the prediction task at hand, despite the underlying context.

2.1 *Prediction of Smartphone Engagement*

First, a distinction between measuring user engagement in terms of attention-based engagement and in terms of usage patterns has to be made (Lalmas et al., 2014). While the first one is hard to measure and can only be captured using self-report surveys or physiological measures, the latter can easily be recorded automatically. Due to this nature, the latter is more suited for machine-learning approaches which rely on the availability of vast amounts of data and is therefore the choice for the present work. Nevertheless, Mathur et al. (2016), who follow the opposite direction by using the EEG signals of users to derive their engagement-scores for mobile phone sessions, deliver an important starting point for predicting app-engagement. By inferring binary engagement labels from EEG signals, which are then used as targets for training an SVM classifier on contextual features, they reach an F1-score of 0.82. Since their approach is based on an expensive study design, its biggest weakness is the lack of feasibility to use it for real-world applications, leading to the open question of how users' engagement can be modeled else.
Tian et al. (2021) present a solution for that by deriving the engagement level from the user's dwell time in dependence of the corresponding overall use-duration quantiles of the underlying app category, thus modeling users' engagement easily as a multiclass classification problem. By doing so, they avoid running into the problem of a highly skewed distribution of the target variable, which is common in the area of dwell time prediction. They use a boosting-based model that is capable of jointly predicting a user's next app choice, and the expected engagement level for it, reaching an accuracy of 0.485. While they find that historical patterns are most important for predicting the engagement level, they do not utilize a model which is capable of remembering historic usage patterns on its own but provide this information to the model in the form of additional features.
As with Mathur et al. (2016), a shortcoming of their work is that they do not treat the user's app events individually but aggregate the dwell time for similar apps within one smartphone session. Additionally, neither of them discusses how to deal with the cold-start problem.
The discussion of the small scope of existing studies shows that there is still a research gap in the field of app-engagement prediction. In particular, the recent prominence of RNNs in the field of next-app prediction can give valuable insights into how to adapt a deep-learning approach for the present work, one that is capable of fully utilizing the predictive power of historical app usage sequences.

2.2 *Next-App Prediction for Smartphones*

Whereas early research in the area of next app prediction first focused on classical machine-learning models like Bayes classifiers (Shin, Hong, & Dey, 2012) or Markov models, (Zou, Zhang, Li, & Pan, 2013) trained on mainly contextual features, more recent research started leveraging the ability of RNNs to capture long term dependencies in app sequences. By doing so, the recorded app history of a user is treated as a sequence consisting of individual app events where the app event at timestep $t + 1$ is assumed to be influenced by the app events at previous timesteps.
Lee et al. (2019) investigated the performance of different RNN architectures for next-app prediction of dual-display devices in comparison with non-deep-learning models. The fact that RNNs, LSTMs, and GRUs outperform standard models supports the general idea of treating prediction tasks related to app events as sequential problems. While a stacked LSTM works best for their prediction task, with an accuracy of 0.75 for predicting the Top-3 apps, they find that adding contextual features to the app history is not crucial for the predictive performance of the models.
In contrast, Xu et al. (2020) detected that contextual features related to time and location can leverage the predictive performance in a similar area. By training an individual LSTM for each user, they reached a precision of 0.8. According to the conflicting results about feature importance, the current state of the literature does not provide an answer about which features should be used or omitted for predicting app usage. Since the studies in the field of app-engagement prediction also did not investigate this in detail, it remains an open research question that will be addressed in the present work to get more clarity.
A shortcoming of both studies is that they do not mention how to deal with the cold-start problem, while they treated the prediction task as user-specific rather than generic. In contrast, Chen, Maekawa, Amagata, and Hara (2021) propose a solution for the cold-start problem by creating user-specific training datasets from a pool of source users. Using a two-layer LSTM, they reached an accuracy of 0.56. Although this approach can be used to make predictions for new users, it is computationally expensive since it relies on individually trained models rather than on a fully generic model.
Instead, Katsarou et al. (2022) present an approach based on a generic LSTM trained on all of the combined user data. By using only phone-history related features to reduce privacy concerns, they reached a recall of 0.76 for the cold-start problem on the 500 most frequently used apps. To utilize information which can be found in the relationships between different applications, they incorporated an embeddings approach where

each app name is converted into a vector in space before feeding it into the LSTM. Since a similar method will be adapted for the present work, entity embeddings will be described in more detail in section 3.
Although several studies compared different RNN architectures and feature subsets, a comparison of RNNs with standard FFNNs is still missing, thus making it hard to determine how much predictive accuracy can be added by using more complex structures. Therefore, the present work will compare the use of an RRN with an MLP to provide an answer to this question.

2.3 *Dwell Time Prediction for Streaming Services*

To further explore the difficulties which can be encountered by performing a dwell time prediction task, a quick look into the area of predicting engagement for streaming services provides useful information.
Vasiloudis, Vahabi, Kravitz, and Rashkov (2017) conducted the first research related to the prediction of session-length in music streaming services. While treating the task as a regression problem with gradient-boosting trees, they emphasized the difficulties arising from a highly right-skewed distribution, which is common in the field of dwell time prediction. Additionally, they dealt with problems related to the huge influence of external factors in the context of a user's session length. Since the session length is determined by factors which cannot easily be recorded, building a prediction model is a challenging task which needs a careful consideration with regard to the features used. To address the problem, they relied mainly on contextual and user-related features.
In contrast, Wu, Rizoiu, and Xie (2018), who conducted the first large-scale study on video watch time, used features related to the consumed content for predicting the average watch-time percentage for videos. Similar to the research in the area of next-app predictions, there is no agreement on the question of which features work best for predicting engagement. At the same time finding a suitable feature subset seems in particular important in the field of engagement predictions to counter the huge influence of external factors. Since no agreement on the question of best-performing features could be found in the literature, several feature subsets, including contextual, user-related, and phone history-related features, will be tested in the present work. As already mentioned, the approach used by Tian et al. (2021) for converting the numerical dwell time to a multiclass target will be adapted for the present work since it avoids the problems of a highly skewed distribution of the target variable.

3 METHODOLOGY

The following section describes and supports the methodology chosen to answer the proposed research questions. First, different neural networks will briefly be described and compared. Second, the proposed LSTM structure will be explained in more detail. Subsequently, the entity embeddings method chosen for treating categorical features will be introduced. Last, an overview of the implemented baselines will be given.

3.1 *Models: Comparison of MLP, RNN, LSTM, and GRU*

A MLP is known as the simplest FFNN where an input, X, is processed through a set of fully connected nodes with the goal of approximating a function, f, used for mapping X to an output, y (Goodfellow, Bengio, & Courville, 2016). Since it has no feedback connections, where the output is fed back to the model as additional input, its main disadvantage is the assumption of independent input datapoints, which is a particular limitation when dealing with sequential data (Goodfellow et al., 2016).
To address this limitation, the concept of RNNs was introduced (Rumelhart, Hinton, & Williams, 1986). In contrast to MLPs, RNNs can refer back to the full history of already-processed inputs for mapping a new input to an output (Graves, 2013). To allow for this, RNNs share the same parameters across several timesteps instead of having separate weights for each input feature. The use of timesteps does not imply that the underlying data has to be in form of classical time-series data with equally spaced units; RNNs can also be trained by processing sequential data where only the position of a datapoint in a sequence, but not the time dimension as such, is important (Goodfellow et al., 2016). Looking at the prediction task at hand, where the data consists of app sequences, the latter one is the case. The most crucial disadvantage of a classical RNN is that, in practice, the historical context that can be leveraged for predicting is quite limited due to a problem known as the vanishing or exploding gradient. This problem arises when the error in an RNN is backpropagated through long sequences leading to either exponentially increasing or decreasing weights.
LSTMs, which were first introduced by Hochreiter and Schmidhuber (1997), are a specific version of RNNs developed to deal with this problem. By making use of a special gate structure, LSTMs regulate the information flow of the network in a way that allows for only memorizing important information while also forgetting information which is no longer needed (Graves, 2013).
The LSTM and its different gates work as follows: First, the forget gate, f_t, decides which information from the previous cell state should be dismissed

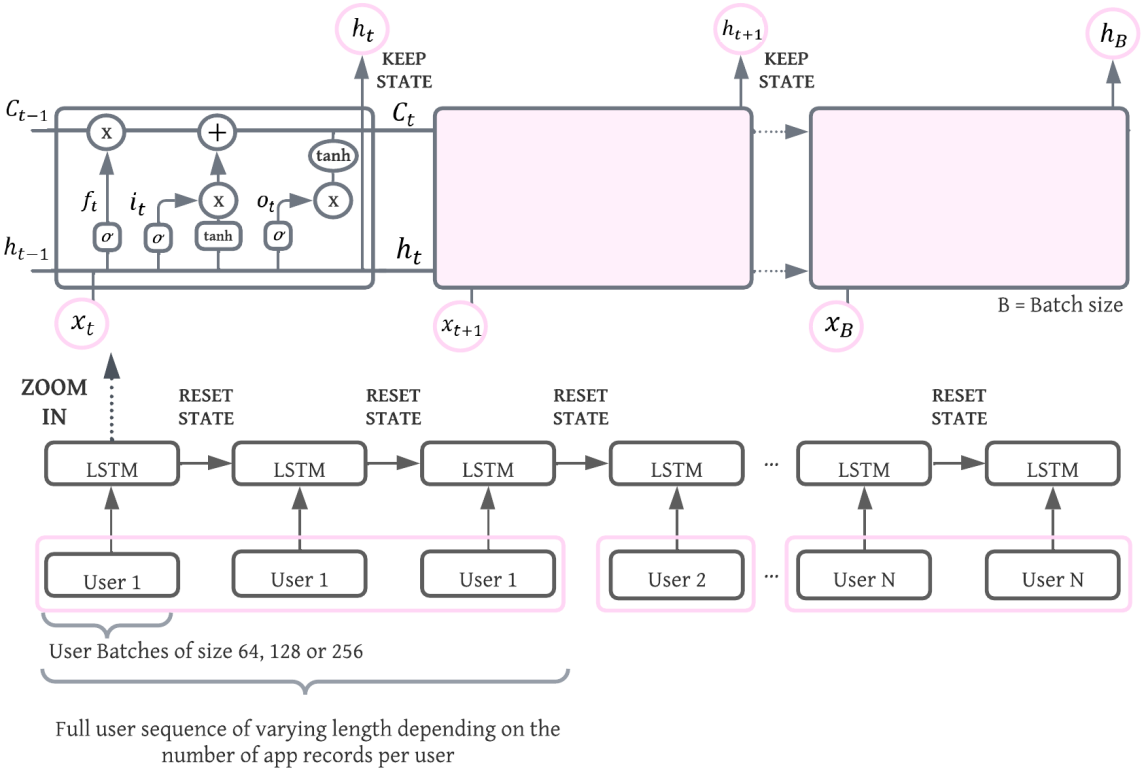

Figure 1: Functional scheme of the proposed LSTM architecture trained on equally-sized batches of sequential user data. Source: the author's illustration based on Christopher Olah (2015)

by processing the data from the current input, h_t, and the output from the previous unit, C_{t-1}, through a sigmoid layer, σ. Next, the input gate, i_t, decides which data in the cell state should be updated and which new data should be added. Subsequently, the cell state, $\tilde{C}_t$, is updated accordingly. Last, the output gate, o_t, decides which parts of the cell state will be used for the output, h_t, and generates it accordingly. The zoom-in in Figure 1 shows a graphical illustration of the workflow of the LSTM. A mathematical formulation can be found in Appendix B.

An alternative for LSTMs is the use of GRUs, which were first developed by Cho, van Merrienboer, Bahdanau, and Bengio (2014). They make use of a similar structure but only consist of two instead of three gates making them more computationally efficient. At this time, the scientific community cannot state with certainty which structure is better suited for dealing with long-term dependencies since this is mostly reliant on the underlying data (Cahuantzi, Chen, & Güttel, 2021; Lendave, 2021; Yang, Yu, & Zhou, 2020). While LSTMs are known for achieving greater accuracy with a sufficient amount of training data, GRUs are the preferred choice when either not

enough data is available or faster model training is necessary.
Since Lee et al. (2019) have shown that LSTMs work slightly better for a related task, the present study has chosen to use them as well. Additionally, an MLP was chosen to investigate whether the more complex structure of an LSTM, which utilizes a sequential version of the data, can reach a higher performance compared to an FFNN trained on a non-sequential version of the data. The choice of the MLP is motivated by the fact that it is the simplest FFNN architecture thus offering a interesting baseline performance. The comparison is thereby particularly important to find a good trade-off between accuracy and computational efficiency, leading to a saving of resources where appropriate.

3.2 *Proposed LSTM Structure - Stateful vs. Stateless Training*

By default, an LSTM can be seen as stateful as it carries the internal cell state between the different LSTM units that process the sequential inputs one by one. From a theoretical viewpoint, this means that a fully stateful LSTM assumes that each step in the input sequence could have possibly been influenced by one of the previous steps (Brownlee, 2020b). While training an LSTM, it is common to avoid feeding it the entire input sequence at one time, but instead split the sequence into smaller sub-sequences which are then processed incrementally (Goodfellow et al., 2016). Processing the data in so-called mini-batches can be used to reset the internal state of the LSTM after each batch, preventing it's current state from being carried over to the next batch (Keras, n.d.-b). By doing so, each batch of data is treated independently from the previous one, which makes it possible to process multiple independent sequences with the same model. Especially in areas where either not enough input data, not enough computational power, or time is available to train one model for each individual sequence, it is crucial to have the possibility to include all combined sequences in one model.
Since the present work aims to construct a generic model trained on a combination of different user sequences, it will utilize the concept of batch processing for training an LSTM on user batches where the state of the LSTM is reset before moving to the next user. While this approach allows us to keep the model structure simple, several things must be considered. First, training the model on full user batches with a varying number of records per users will lead to a model that is potentially at risk of biasing the predictions towards users for whom more training data is available. To counter this risk, it was decided to train the model not on full user sequences but on equally sized batches of user data. By further splitting the full user sequences into smaller sub sequences, the model will be prevented

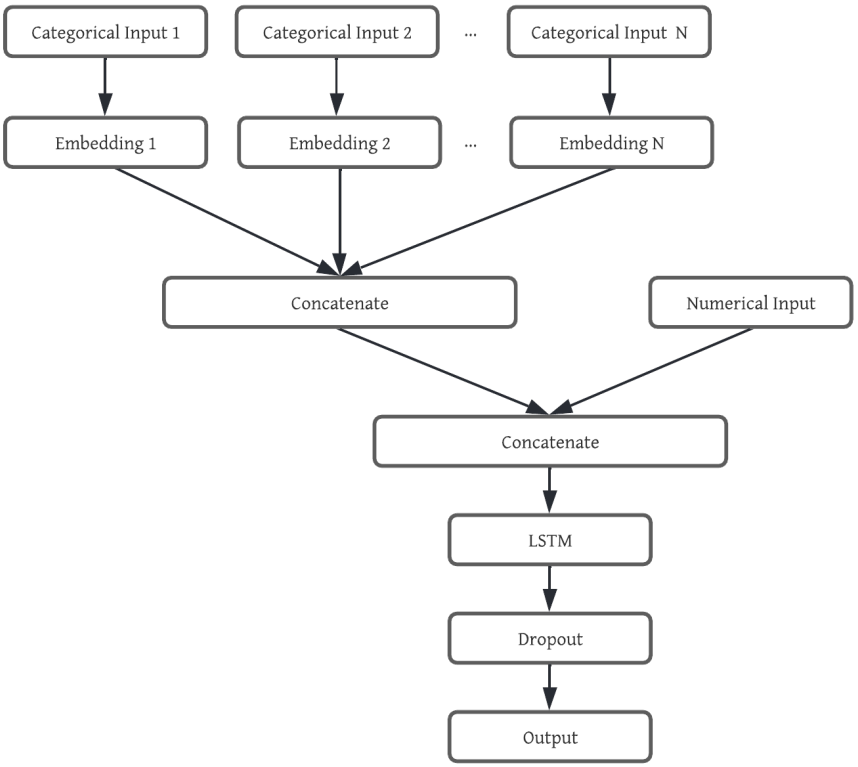

Figure 2: Illustration of the implementation of the proposed LSTM architecture with an embeddings and dropout layer. Source: the author's illustration based on a figure which was created in Python by using Keras (see Appendix C)

from memorizing extremely long user sequences in total. Since the internal memory state of the model will be reset after finishing the training on one batch, all batches will be treated as individuals even if they originate from the same user.
An illustration of the proposed model structure is given in Figure 1. What cannot be seen in this illustration is that, additionally, the order of the user batches will be shuffled before processing them with the LSTM.

3.3 *Treatment of Categorical Features - Embeddings Method*

After deciding on the general model structure, there are several possible ways of treating categorical features during training. While traditional approaches mostly rely on a one-hot encoding of features, resulting in the disadvantage of getting a sparse representation without depicting the

relationship between different values of a category, the approach of entity embeddings preserves the semantic relationships of the categories' values while also transforming them into a lower-dimensional space (Guo & Berkhahn, 2016).
Embeddings map each value of a categorical feature to a vector representation where the number of dimensions can be freely chosen. By doing so, semantically more similar values are grouped together in space, whereas dissimilar values are kept apart. Since the embeddings will be used as a basis for training the MLP and LSTM, they can simply be added to those models by prepending an embeddings layer to their general structure. By using the Keras Embeddings Layer, the embeddings for each variable are learned from the scratch as a part of training the whole model (Keras, n.d.-a).
The embeddings size of each variable was chosen, based on the rule proposed by Howard and Gugger (2020):

$$\textit{Embeddings size} = \textit{cardinality size}/2 \quad \textit{but no bigger than } 50 \tag{1}$$

Figure 2 illustrates the implementation of the described LSTM architecture with all its components.

3.4 *Baselines*

First, a simple majority vote will be used to compare the results to a non-learning model. Second, a linear model, namely a ridge classifier, will be used as an additional baseline. This model was chosen because it is well suited for handling the high dimensionality of the input data. Last, an MLP trained on a one-hot encoded version of the categorical data is included to allow for insights into the effect of the implemented embeddings layer. Those baselines will be implemented for the cold-start problem separately, with an exception of the MLP without embeddings.

4 EXPERIMENTAL SETUP

This section will give a detailed overview of the experimental setup of the present study. Figure 3 illustrates the workflow graphically. More detailed graphics will be presented in the different sections to give in-depth insights into the chosen methodology.

4.1 *Description of the Dataset*

The selected dataset was collected as a part of a study, conducted at Tilburg University, to investigate the relationship between phone usage, mental

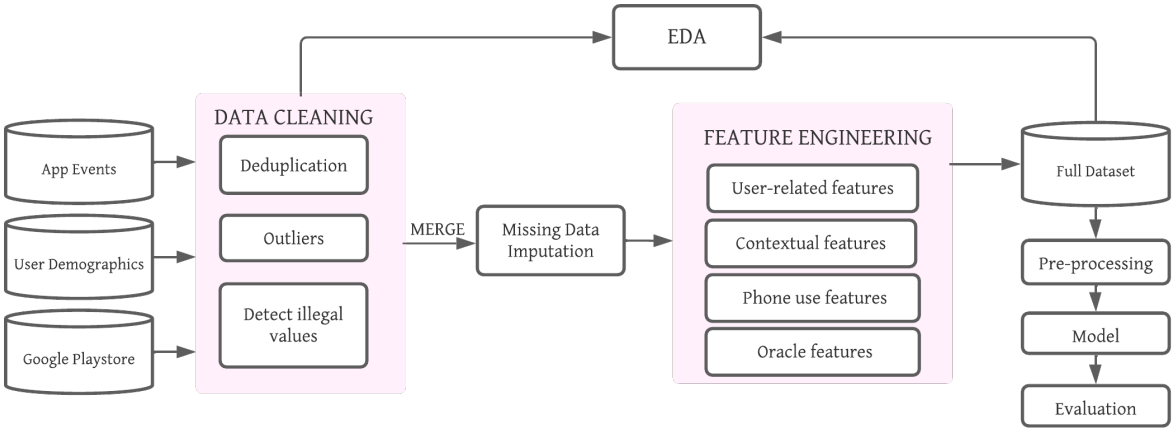

Figure 3: Illustration of the general research workflow focused on data cleaning and feature engineering. Source: the author's illustration

health, and academic achievement among students. For this purpose, the app events of volunteering students were recorded in a time period between 01-23-2020 and 06-20-2020 using a passive logging app called MobileDNA. In addition, the participants were asked to fill in mood surveys (see Aalbers, Keijsers, Abeele, and Hendrickson (2020) for a detailed description of the study design and data collection). For the purpose of the present work, only data related to phone usage and demographics will be used.
Before cleaning, the data sets consist of a total of 5,824,291 app logs from 272 participants. Each app-log includes a unique user ID, an identifier of the used application, the start and the end time of the use, the battery level of the phone, the location, and whether the use was initiated by a notification. The demographics include the age and the sex of each participant.
Following the approach used by Tian et al. (2021) for constructing the target variable, information about the app categories is needed (see section 4.5 for details). Since those were not included in the original dataset, an additional dataset containing information regarding each app's assigned category in the Google Playstore was retrieved from Kaggle (Prakash, 2021). To provide some insight into the properties of the datasets, Appendix D shows some interesting results from the EDA.

4.2 *Data Cleaning*

First, all datasets were deduplicated and a search for illegal values was conducted. One illegal age value was detected and removed for a later imputation. One app event with a negative use duration was dropped.

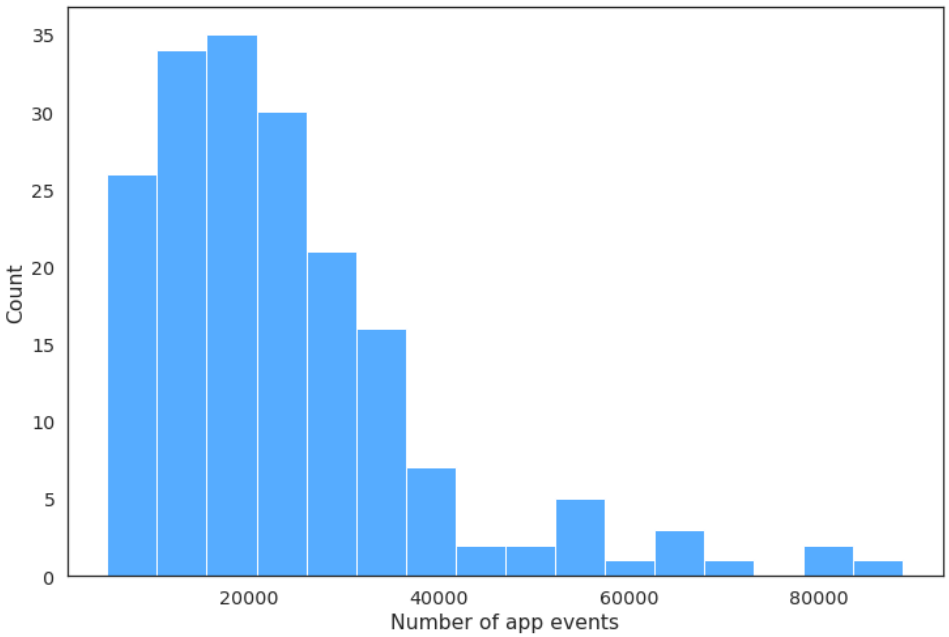

Figure 4: Histogram depicting the distribution of the number of app events per user after data cleaning. Source: the author's illustration

From a statistical viewpoint, there are several outliers in the feature use duration. As already discussed, one main challenge for predicting the dwell time is the underlying, highly right-skewed distribution. Since the prediction task will be addressed by making it a multiclass classification problem, and the target construction is not highly affected by the outliers, they will be kept as they are.
Users with less than 5,000 app events were dropped to ensure that the planned user-based train-validation-test split (see section 4.6 for details) leads to subsets large enough to provide meaningful results. Due to that restriction, the initial number of users was reduced to 186 with a total of 4,419,918 app records, whereas the number of records per person varies between 5,013 and 88,895. Figure 4 shows a histogram of the distribution of the number of app events per user. After the initial cleaning, all datasets were merged together to treat missing data.

4.3 *Missing Data Treatment*

Besides the dropped illegal age value, only latitude and longitude contain 1,114,792 missing values. It has been shown from previous studies that the location of the users contains valuable information about their phone

Categorization	Feature	Description	Type
User-related	Age category	Age group of the user: 4 categories	Cat
	Sex	Sex of the user: male and female	Cat
Contextual	Battery level	Battery level of the phone: 0-100	Cat
	Hour of the day	Different hours of a day: 0-23	Cat
	Weekday	Day of the week: Monday to Sunday	Cat
	Location cluster	Assigned location cluster: 0-7	Cat
	Geohash	Geohash of the location: 1837 categories	Cat
Phone history	Use duration	Use duration of the app in seconds	Num
	Duration ongoing session	Duration of ongoing session in seconds	Num
	Time since last app event	Time since last app event in seconds	Num
	App category	Category of the used app: 32 categories	Cat
	Notification	If app event was initiated by a notification: True or False	Cat
	Engagement level	Engagement level of app event: short, medium, long	Cat
Oracle	Category of next app	Category of the next app: 32 categories	Cat
	Time to next app	Time to next app in seconds	Num

Figure 5: Full list of features, including categorization, description, and type of feature where constructed ones are highlighted in light pink. Source: the author's illustration

usage behavior (Xu et al., 2020). Thus it was decided to adapt a multiple imputation strategy to avoid losing useful information.

4.4 *Feature Engineering and Transformation*

Based on the already existing features, several new ones were constructed. Following the domain literature (Tian et al., 2021), features were categorized into user-related, contextual, phone history related, and oracle features which already provide information about the app events for which the engagement class has to be predicted. A detailed overview of all features and their categorization can be found in Figure 5.

4.4.1 *User-Related Features*

The age variable was used to create four age categories, based on the quantiles.

4.4.2 *Contextual Features*

Based on the start time of each app-event, two time-related features were constructed: the weekday and the hour of a day. To utilize location as a feature, latitude and longitude were transformed into geohashes. Geohashes are unique identifiers that encode latitude and longitude values into short strings representing a specific region of the world (Hill, 2017).

The number of characters of the geohash is dependent upon the size of all individual regions, wherein longer hashes lead to smaller regions. As a trade-off between precision and the number of distinct categories, the length was set to five. Additionally, a k-means algorithm was used to cluster the dataset based on the latitude and longitude values into eight groups.

4.4.3 *App-History Related Features*

Based on the time between app events, usage sessions were constructed. Following Tian et al. (2021), a session consists of app events with no more than five minutes in-between. The duration of an ongoing session, at the moment of starting an app event, was calculated and added as a feature. Additionally, the duration between an app-event and the previous app-event was calculated and added as feature.

4.4.4 *Oracle Features*

Two oracle features were added to the dataset. First, the category of the app at the next timestep, and second, the duration until the app at the next timestep is opened were added to each app record.

4.4.5 *Feature Transformation*

All numerical features were standardized by using the StandardScaler from sklearn to improve the performance of all neural networks (Anysz, Zbiciak, & Ibadov, 2016; Shanker, Hu, & Hung, 1996). For the Ridge Classifier and the simple MLP baseline, categorical features were one-hot encoded. For all other models they were treated by an embeddings layer.

4.5 *Construction of Target*

The prediction task will be handled as a multiclass classification problem with three output classes, short, medium, and long. Following the approach of Tian et al. (2021), the engagement level of an app will not be solely based on its use duration, but constructed with regard to the corresponding app category and its 33% and 67% quantiles for use duration. This transformation solves the problem of a highly skewed numerical outcome variable, which is often encountered during dwell time prediction tasks and can be seen in Figure 6. Additionally, it results in a balanced target.

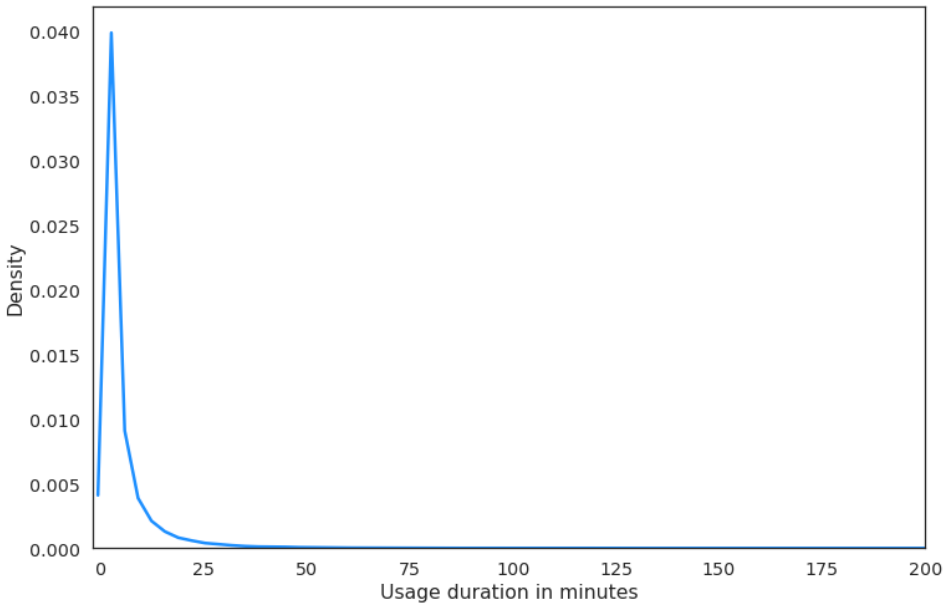

Figure 6: Kernel density estimation plot of the usage duration in minutes. Source: the author's illustration obtained using Python

4.6 *Training-Validation-Testing Split*

The full dataset was split into three subsets. Since the LSTM will utilize the sequential character of the underlying data, a random split of the records is not appropriate. Instead, the app records of each individual user were split in such a way that approximately the first 80% of the records were used for training, the next 10% for validation, and the last 10% for testing. Thus, each subset contains records of the same users, only in a different time periods. To ensure comparability between models, the same method of splitting was adapted for the MLPs and baselines.
To test the model's performance for the cold-start problem, a different split was chosen, where 80% of the users with their full records are randomly chosen to be in the training set, 10% in the validation set, and 10% in the testing set, resulting in subsets without any user intersection. For a better understanding, Figure 7 provides an illustration of the pre-processing for all different models and tasks.

4.7 *Data Pre-processing for LSTM and MLP*

Since the two models will treat the task differently, they need a separate pre-processing and modeling of the data structure used for model training. As already explained, the MLP only takes into account the features of

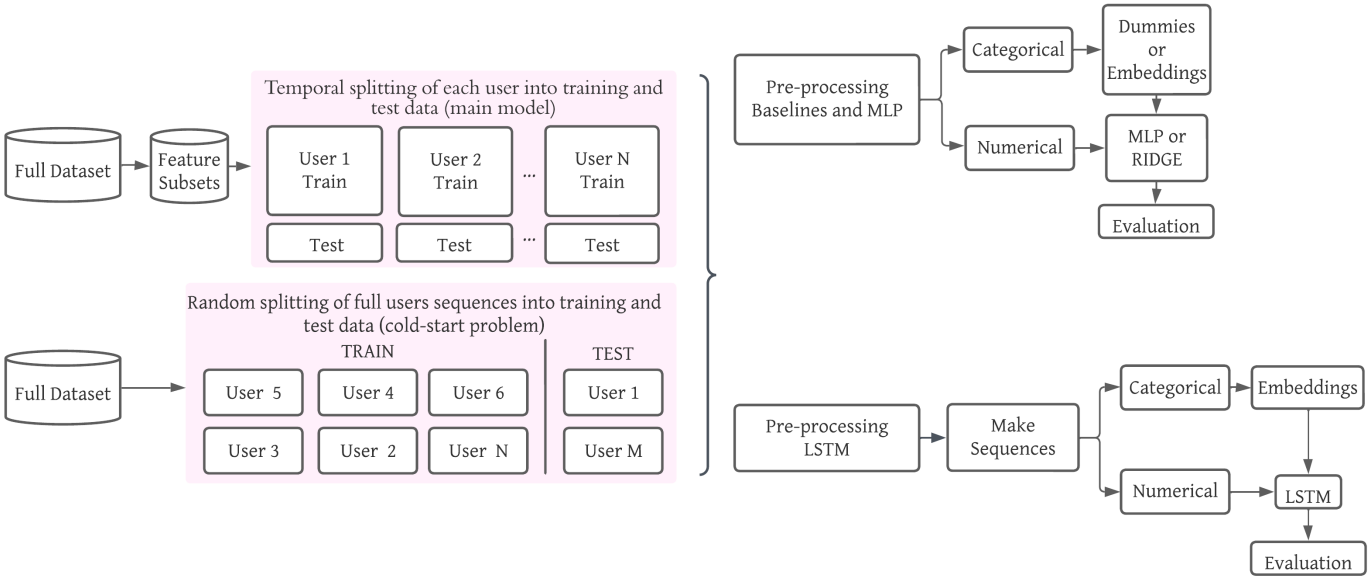

Figure 7: Workflow focused on the training-validation-testing split and pre-processing for all baselines, the MLP, and the LSTM on the main task and the cold-start problem. Source: the author's illustration

one app record to predict the dwell time for the following app event. Accordingly, the features at timestep $t-1$ are utilized for predicting the target of the timestep t.
Instead, the LSTM will use a sequences of previous app records to predict the following one. To transform the structure in a way which allows for that, a sliding window method was used (Brownlee, 2020a). The number of previous app records that are taken into account for predicting the next apps dwell time is defined as hyperparameter and will be referred to as the lookback range. Current literature flags it as one of the most important hyperparameters for similar prediction tasks (Katsarou et al., 2022; Xu et al., 2020). Figure 8 gives an illustration of the different data structures.

4.8 *Hyperparameter Tuning*

All neural networks will be optimized using an out-of-sample evaluation method, where the models will first be trained on the training set. Next, the hyperparameter settings will be chosen on the validation set. Last, the predictive accuracy will be obtained from the test set.
The most important hyperparameter for the proposed LSTM is the batch size. While a larger batch size would allow for a more specific pattern

MLP: Data structure used for training

	Input	Output
Example 1	Features t-1	Target t
Example 2	Features t	Target t+1
Example 3	Features t+1	Target t+2
...	...	...

LSTM: Data structure used for training

	Input	Output
Example 1	Features t - 5 Features t - 4 Features t - 3 Features t - 2 Features t - 1	Target t
Example 2	Features t - 4 Features t - 3 Features t - 2 Features t - 1 Features t	Target t+1
Example 3	Features t - 3 Features t - 2 Features t - 1 Features t Features t +1	Target t+2
...	...	...

Figure 8: Exemplary illustration of the differences in the data structure used for training the MLP (left) and the LSTM (right). Source: the author's illustration

recognition in each individual user sequence, a smaller batch size is expected to lead to greater generalizability. At the same time, the batch size is dependent on the shortest available record. After splitting the dataset, the shortest user sequence in the test set has a length of 501. Therefore, it was decided to choose 256 as maximum batch size to test.
To allow a training of the LSTM on sequential user batches, each user subset must be dividable by the chosen batch size. Accordingly, all users' app-records are truncated to a number dividable by 256 to allow for testing batch sizes of 64, 128, and 256 as hyperparameters. This procedure was chosen to avoid the need for additional computational power and memory space resulting from zero padding. Since only about 0.5% of the dataset was lost, no negative effects are expected.
By implementing a manual optimization loop, first the batch size, in combination with the number of neurons that decide upon the complexity of a model, is optimized for both the MLP and the LSTM. Therefore, all possible combinations of the values 64, 128, and 256 will be tested. Since Greff, Srivastava, Koutník, Steunebrink, and Schmidhuber (2017) have shown that it is reasonable to tune the hyperparameters of an LSTM independently, the lookback range of the LSTM, the second most important hyperparameter, will only be tuned afterwards by testing the values 5, 10, and 20.
To find the best number of training epochs, and prevent the models from overfitting, an early stopping mechanism with a patience value of 5 is used during optimization. After stopping, the best model weights will be restored.

For both models an ADAM algorithm is used for gradient-based optimization and categorical cross-entropy is used as the loss function. For the hidden layer, sigmoid, and for the output layer, softmax, are used as the activation functions. To prevent the models from overfitting, a dropout layer is added with a dropout rate of 0.2. Those choices were all based on relevant, previously conducted research (Katsarou et al., 2022; Xu et al., 2020).

4.9 *Robustness Check*

The robustness of the MLP and LSTM is checked by resampling the data used for training in-between three independent training periods. Since a complete resampling of the data is not possible, because of the sequential structure of the data, the order in which the records are fed into the models is randomized and resampled. In the case of the LSTM, this takes place on the level of the user batches; in case of the MLP, on the level of the individual records.

4.10 *Evaluation Metrics*

Following the already existing literature in the context of smartphone engagement (Tian et al., 2021), the chosen evaluation metric is accuracy. Since the target was modeled in a way resulting in balanced output classes, it seems appropriate for evaluation.

4.11 *Software*

All steps were completed using the programming language Python 3.0 in a Juypter Notebook. The following packages were used:

- Pandas 1.0.5 (McKinney, 2010)
- NumPy 1.22.3 (Harris et al., 2020)
- Fancyimpute 0.7.0 (Rubinsteyn & Feldman, 2016)
- Sklearn 1.0.2 (Pedregosa et al., 2011)
- Pygeohash 1.2.0 (McGinnis, 2015)
- Seaborn 0.11.2 (Waskom, 2021) and Matplotlib 3.5.1 (Hunter, 2007)
- Keras 2.8.0 (Chollet et al., 2015) with TensorFlow 2.8.0 (Abadi et al., 2015) as backend

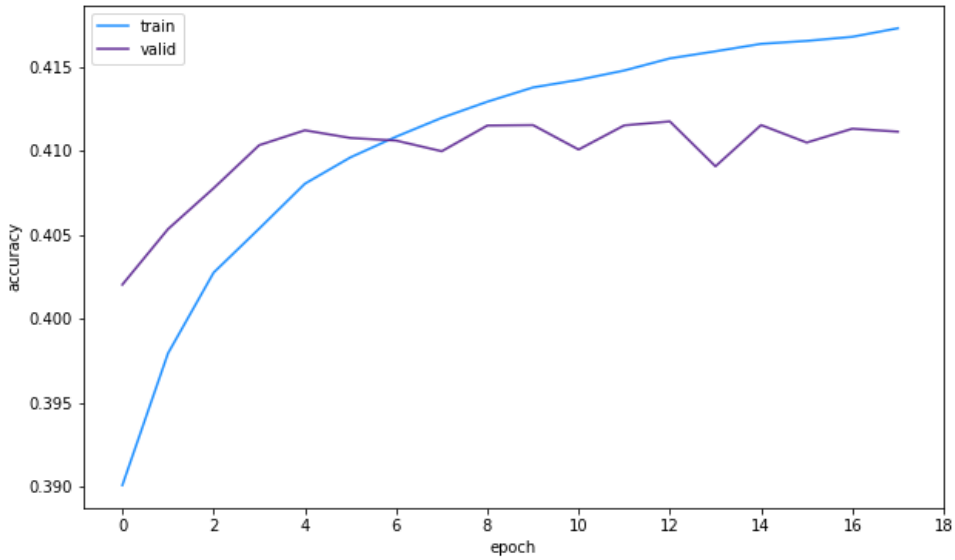

Figure 9: Accuracy plot of the training phase of the MLP on the full feature subset by the use of the best hyperparameters showing training and validation accuracy in comparison. Source: the author's illustration obtained using Python

5 RESULTS

The following section will present the results for all baseline models, the MLP, and the proposed LSTM. To begin, the hyperparameter optimization for all models is unfolded. After that, and by following the structure of the research questions, first, the performance of the LSTM and the MLP on the full feature subset will be compared to the baselines, along with the results of the robustness check and error analysis. Second, the results of training the MLP and LSTM on different feature subsets will be shown. Last, the performance of the LSTM on the cold-start problem will be evaluated.

5.1 *Hyperparameter Tuning of the Neural Networks*

To begin, Figure 9 illustrates the early-stopping procedure used to determine the best number of epochs by showing one exemplary accuracy plot that was obtained while training the MLP on the full feature subset. By looking at this figure, it can be seen that the optimal validation accuracy for this model could be obtained at epoch 12.
Table 1 shows an overview of the best hyperparameters for the MLP and the LSTM on all different feature subsets. Table 2 shows the best hyperparameters for the MLP without embeddings, and the models for the cold-start problem. Appendix E includes all of the scores that were used to

	Hyperparameters	
	MLP with Emd.	LSTM
full subset	B: 256 N: 256	B: 256 N: 256 L: 20
full subset incl. oracle	B: 256 N: 256	B: 256 N: 256 L: 20
contextual + history	B: 256 N: 128	B: 256 N: 128 L: 20
only history	B: 256 N: 256	B: 128 N: 256 L: 20

Table 1: Best hyperparameters for the MLPs with Embeddings and the LSTMs on different feature subsets where B denotes the batch size, N the number of neurons, and L the lookback range

	Hyperparameters		
	MLP no Emd.	MLP cold-start	LSTM cold-start
full subset	B: 128 N: 128	B: 64 N: 256	B: 64 N: 256 L: 20

Table 2: Best hyperparameters for the MLP without embeddings and the models for the cold-start problem on the full feature subset, where B denotes the batch size, N the number of neurons, and L the lookback range

produce both tables. Most models worked best with a higher number of neurons, indicating that more complex structures can better capture the patterns of the underlying data. For most of the models, a greater batch size worked best, with the exception of all the models which were used to address the cold-start problem. For those models, the smallest batch size generally worked best, which supports the assumption that smaller batch sizes are advantageous for a better generalization while larger batch sizes lead to a closer fit to the data. Additionally, it has been found that the performance of the LSTM constantly increased by increasing the lookback range. Since the memory of the machine used for this study was limited to 256GB, we could not test whether any further increase would continue this trend.

	Accuracy		
	train	valid	test
Majority Vote	0.3410	0.3340	0.3376
Ridge Classifier	0.3723	0.3730	0.3700
MLP no Emd.	0.3934	0.3897	0.3851
MLP with Emd.	0.4195	0.4120	0.4061
LSTM	**0.4389**	**0.4296**	**0.4262**

Table 3: Results of all baseline models, the MLP, and the LSTM trained on the full feature subset with the best hyperparameters on the training, validation, and testing sets

5.2 *Performance of the Baselines, the MLP, and the LSTM on the full feature subset*

To begin, Table 3 gives an overview of the performance of all baselines, the MLP, and the LSTM on the full feature subset. While the majority vote resulted in an accuracy of 0.338, the Ridge Classifier obtained a slightly better result, increasing the accuracy to 0.37.
All proposed neural networks outperformed the linear baselines. The results of the MLP with embeddings layer and the LSTM were obtained after averaging the results over three training periods. By comparing the performance of both models during the different training phases, it can be seen that they reached a relatively stable performance across the re-sampled training sets which is an indication of the model's robustness. A detailed overview of the results from the robustness check can be found in Appendix F.
The LSTM has a tendency to overfit the training data, while the MLP is less at risk of overfitting, which result can be seen by looking at the accuracy plots obtained during training (see Appendix G), as well as the accuracy differences on the training and validation set.
While the MLP, trained on a dummy-encoded version of the dataset, reached an accuracy of 0.385, the MLP which utilized entity embeddings could further improve the performance by 0.021 to 0.406. The LSTM outperformed the simple MLP by 0.02, reaching an accuracy of 0.426, thus being the best performing model.

5.3 *Error Analysis*

To begin, Figure 10 shows a comparison of the multiclass confusion matrices derived from the LSTM's and MLP's predictions on the test data using

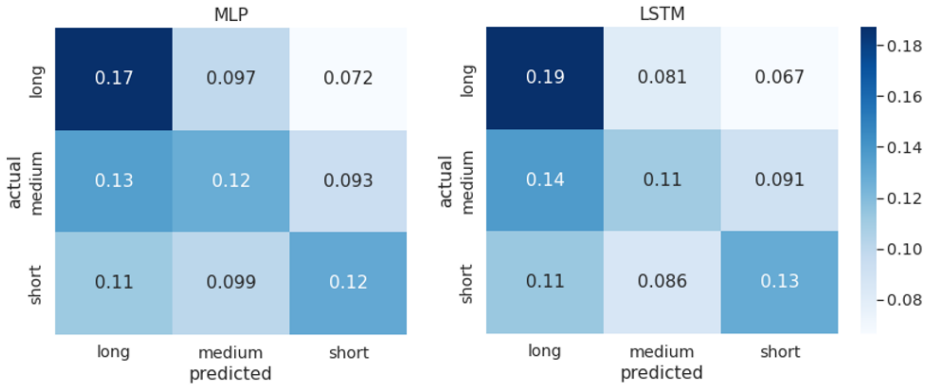

Figure 10: Multiclass confusion matrices of the MLP (left) and the LSTM (right) obtained from the predictions on the test set by the models trained on the full feature subset using the best hyperparameters. Source: the author's illustration obtained using Python

the full feature subset. Based on that, it can be seen that both models work best for predicting long-duration app usage. While the LSTM performed worst for the medium class, the MLP is slightly better, reaching the same accuracy for the medium and short class. The LSTM performed slightly better on the short-duration class than the MLP. The performance differences between both models can thus be entirely explained by the LSTM's better performance on the long-duration class. In general both models have a bias towards predicting the long-duration class.
To check for a bias between users with a varying amount of available training data, the LSTM's predictive performance for all users was compared. Figure 11 gives an illustration of the results. The length of a user's training data can be found on the x-axis, while the y-axis shows the accuracy. The purple line was obtained by a linear regression, the pink line was obtained by using a robust regression technique, namely Huber regression, and thus accounts for outliers. As can be derived from the plot, the model has a small bias towards those users for whom more training data is available. To quantify it, the coefficient of the Huber regression line is $2.66e^{-07}$.

5.4 *Performance of the MLP and the LSTM on different Feature Subsets*

Both the MLP and the LSTM were tested on different feature subsets. Table 4 shows an overview of the results obtained. As can be seen, the inclusion of the created oracle feature increased the performance of the MLP, compared to the standard feature subset, by 0.035, leading to an accuracy of 0.442. The same applies to the LSTM, for which the inclusion of oracle features improved the performance by 0.035, leading to an accuracy of 0.461.

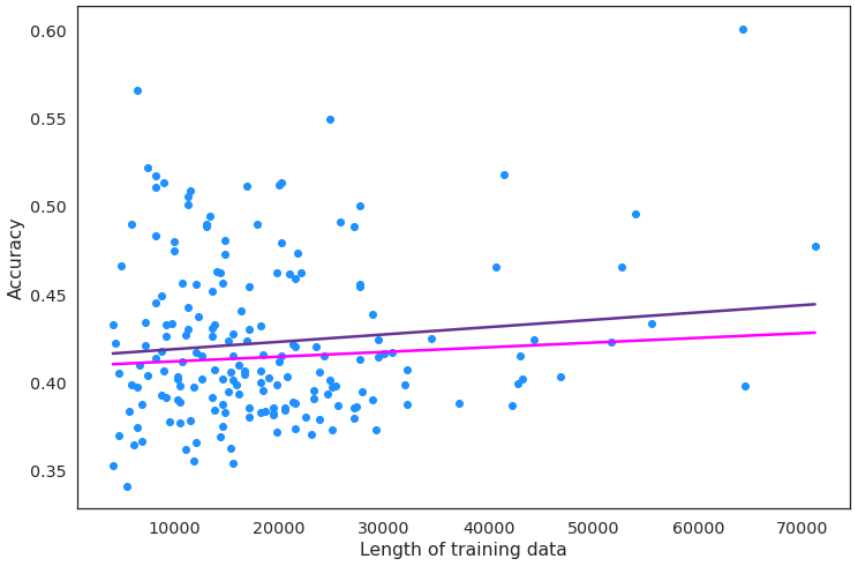

Figure 11: Scatter plot depicting the accuracy of each user on the testing set obtained by the LSTM trained on the full feature subset dependent on the corresponding number of available records in the training data, including a linear regression line (purple) and a Huber regression line (pink). Source: the author's illustration obtained using Python

In contrast, removing user-related and contextual features led to a decrease of the performance for both models. While removing the user-related features only led to a small decrease in performance, 0.006 for the MLP and 0.001 for the LSTM, the loss of contextual features had a bigger impact on the models' performances. The MLP's performance decreases massively by 0.037 from 0.406 to 0.369, whereas the LSTM only lost 0.006, going from an accuracy of 0.426 to 0.420. By comparing the performance of the MLP and the LSTM on all different feature subsets, it is remarkable that the MLP's performance drops more rapidly when removing user-related and contextual features while the LSTM is able to keep the performance more constant.

A comparison of the computing times clearly shows that all MLPs have a shorter training time than the LSTMs, where the training times are dependent on the underlying feature subsets, the batch size, and the number of neurons. Therefore, the LSTM that was trained on contextual and history-related features with a batch size of 256 and 128 neurons had a faster training time than the LSTM trained on only the history, with batch size of 128 and 256 neurons.

	Accuracy		Training time	
	MLP	LSTM	MLP	LSTM
Full subset incl. oracle	**0.4415**	**0.4613**	0:23:11	12:26:37
Contextual and history	0.4005	0.4249	0:17:21	**03:42:02**
Only history	0.3694	0.4201	**0:09:20**	17:06:10
Full subset	0.4061	0.4262	0:20:49	10:33:27

Table 4: Results of the MLP and the LSTM, trained on different feature subsets, using the best hyperparameters, including the training time

5.5 *Performance of the LSTM on the Cold-Start Problem*

As can be seen from Table 5, the MLP's performance on the cold-start problem only improves by 0.003 upon the majority vote, reaching a score of 0.356, whereas it is worse when compared to the performance of the ridge classifier which got an accuracy of 0.365. The LSTM is the best performing model on the cold-start problem and reaches an accuracy of 0.409, thus improving by 0.053 compared to the MLP. Remarkable to see are the differences between the training, validation, and testing sets of the deep learning models. While the MLP loses a total of 0.063, the LSTM loses 0.02 of accuracy between the training and the testing set.
This can mainly be explained by the fact that the models have to generalize to previously unseen users, where the LSTM is more successful in addressing this problem than the MLP. This assumption is strengthened when examining the accuracy plots obtained during training of the models. While we would normally expect the validation accuracy to increase along the epochs until a point of saturation is reached, the Figures in Appendix H show a different observation.

	Accuracy		
	train	valid	test
Majority Vote	0.3411	0.3642	0.3530
Ridge Classifier	0.3724	0.3723	0.3652
MLP with Emd.	0.4195	0.3754	0.3561
LSTM	**0.4290**	**0.4061**	**0.4091**

Table 5: Results of the different models, trained on the full feature subset, using the best hyperparameters for training, validation, and testing data on the cold-start problem

While the accuracy on the training set constantly increases for both models, the validation accuracy stops increasing after the first epochs and drops rapidly afterwards. This pattern can be explained by taking into consideration the data split: Since the data sets were constructed without any user intersection, obtaining a closer fit to the training data, and accordingly to the app usage behavior of the users included in the training data, leads to a loss in the capability of generalizing to the usage patterns of the previously unseen users included in the validation data. This is also in accordance with the observation that a smaller batch size worked better for the models trained for cold-start problem.

6 DISCUSSION

The main goal of the present research was to investigate to what extent a user-independent deep-learning approach can be used to solve the multiclass classification problem of dwell time prediction for mobile phone applications. To give an in-depth answer to the main research question, the results obtained will be analysed from three different perspectives: the performance of the models and their components, the trade-off between accuracy, user-privacy, and computational efficiency, and the capabilities of generalization. Each perspective will be set in the context of the already existing research. Additionally, limitations will be shown and recommendations for future research will be discussed.

6.1 *Predictive Performance of the MLP and LSTM compared to the Baselines*

When looking at Table 3, it can be seen that even the simplest MLP outperformed the linear baselines. It is notable that making the MLP more complex by choosing an embeddings method for encoding categorical features can increase the performance by 0.021. This emphasizes the importance of capturing the relationships between categorical features as it remarkably improves the performance of the MLP which stayed unaltered in all other aspects.
Switching from an MLP to the proposed LSTM, which allows for not only taking into account the last opened app of a user but the dependencies between all previously seen app events in the current batch, further increased the performance by 0.02. Additionally, each increase in the lookback range of the LSTM further increased the performance. This supports the assumption of Lee et al. (2019) that treating prediction problems related to app usage as sequential problems is not only suitable but also advantageous in terms of increasing the predictive accuracy.
Comparing the LSTM's performance to the current state of the art as is

provided by Tian et al. (2021), who reached an accuracy of 0.485, reveals that the present work could not improve upon their performance for two reasons. First, since they treated the problem as joint-prediction task, they already had access to the following app's category which was then utilized for the dwell time prediction. Including so-called oracle features in the feature subset used for training the proposed LSTM increased its performance to 0.461, but it still lagged behind that of Tian et al. (2021), which is where the second reason comes into play. Since Tian et al. (2021) aggregated the dwell time of the applications on a session-based level before creating the target variable, they, in general, dealt with more pronounced differences in their target variable. Working on an application-based level instead means dealing with hardly recognizable differences in dwell time, thus making the prediction task more challenging.

6.2 *Discussion of the Model Performance on different Feature Subsets and the Trade-Off between User-Privacy, Computational Efficiency, and Accuracy*

By training both models on different feature subsets, two remarkable differences could be recognised: First, it is clear to see that the MLP is, as expected, the better choice in terms of computational efficiency. This can be explained by the greater model complexity of the LSTM, but also by the greater complexity of the underlying sequential training data.
In contrast, the LSTM is more suited for keeping the accuracy on a reduced-feature subset high. While the LSTM only loses 0.006 of its accuracy between the full feature subset and the one which only includes the phone history, the MLP loses a total of 0.037. This is most likely based on the fact that the LSTM can leverage the power of the sequential information when features are removed, while the MLP loses information crucial for the predictive accuracy and has no way to recover from this loss. This observation supports the findings of Katsarou et al. (2022) who showed that an LSTM trained on only the phone-history of a user can perform as well as a model which includes additional features on a next-app prediction task, but is also in accordance with the works of Tian et al. (2021) and Mathur et al. (2016) who showed that contextual and user-related features are well suited for predicting dwell time.
The best predictive accuracy could be reached by the models trained on the full-feature subset including oracle features, whereby the LSTM still yields better results than the MLP. Since including oracle features improved the performance by 0.035 for the LSTM and the MLP, the results give an incentive for treating the task of dwell time prediction as a joint prediction task, as Tian et al. (2021) already did in their work. Figure 12 illustrates what these results indicate for the trade-off between user

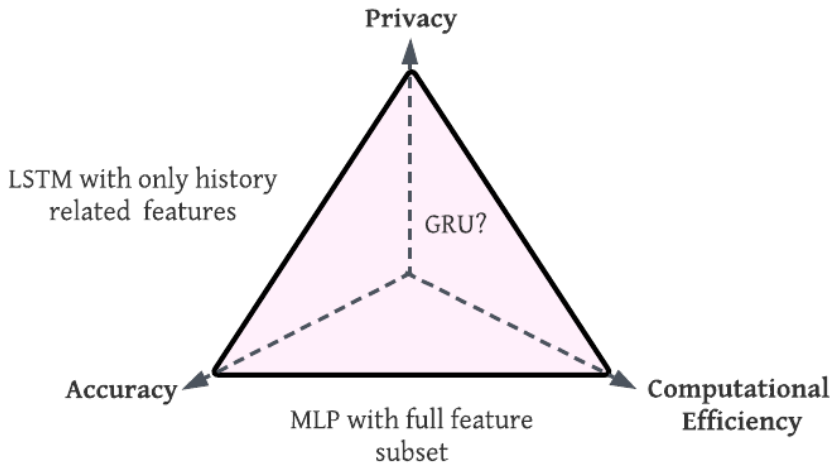

Figure 12: Different deep learning architectures, placed on a magical triangle, illustrating the trade-off between privacy, accuracy, and computational efficiency by taking into account the results obtained. Source: the author's illustration

privacy, computational efficiency, and accuracy. While the MLP offers the best trade-off in terms of computational efficiency and accuracy, the LSTM offers the best between accuracy and privacy. None of the tested models and different feature subsets could offer a good trade-off between all three aspects since the LSTM's computing time is still high on a reduced feature subset, while the MLP's accuracy on the reduced feature subset is not convincing.
One option worth investigating in future research is the use of a GRU model that can offer a trade-off between accuracy and computational efficiency. While it can leverage the power of the sequential structure in a similar way to the LSTM it is, at the same time, less demanding in terms of computational power.

6.3 *Capabilities of Generalization*

The generalizability of the proposed LSTM can be evaluated from two perspectives: generalization to users with less available training data and generalization to previously unseen users.
As could be seen from the error analysis, the proposed LSTM cannot prevent a small bias towards users for whom more training data is available from appearing. Despite that, the proposed model offers an opportunity for including multiple independent sequences into a simple architecture by, at the same time, reaching a relatively stable performance between users.

Additionally, the models' generalization to previously unseen users, which was investigated by looking at the cold-start problem, led to interesting results. The cold-start version of the LSTM reached an accuracy of 0.409. As expected, the batch size of the LSTM, which was chosen as a hyperparameter, plays an important role for finding a trade-off between the generalization and a closer fit which is particularly important for addressing the cold-start problem.
Since the validation accuracy for both deep learning models stopped increasing after one epoch, a main advantage of deep-learning models, the learning process aiming for capturing complex patterns in the underlying data, could not be fully leveraged in the cold-start prediction task. As a result, a simple linear baseline was able to outperform the MLP. What could not be answered with that observation is whether a more complex deep-learning structure would be able to counter this problem, or if a simpler model is indeed advantageous for that particular task. Based on that, future research should definitely take a closer look at the comparison of simple models and more complex deep-learning models on the cold-start problem.
One alternative worth investigating for both generalization tasks could be the use of a transfer-learning approach, which would work on costs of the current models' simplicity.

6.4 *Limitations and Further Recommendations*

Besides the already discussed shortcomings, the proposed research has two main limitations. First, the homogeneity of the dataset used probably produced misleading results. Since the phone usage of members of a homogeneous group is expected to be more similar, compared to a heterogeneous group, all findings related to generalizations between users have to be treated cautiously and must be confirmed on a set of diverse users. The same applies to the observed effect of user-related features on the predictive accuracy of the models.
Second, the limited computational resources did not allow for extensive hyperparameter tuning related to the number of neurons and the lookback range of the LSTM. Furthermore, only a one-layer LSTM could be tested. Accordingly, future research should investigate the influence of a more complex model structure, or different sets of hyperparameters.

6.5 *Contributions and Societal Impact*

The present research made a novel contribution to the field of dwell time prediction for mobile phones by being the first that adapted a deep-learning

approach to leverage the power of the sequential structure of app usage data. By doing so, a generic modeling approach, which is not only capable of including various independent sequences but can also be used to attack the cold-start problem, was introduced. While, from a business perspective in particular, the possibility to increase users' satisfaction by deploying the model to a real-world application is valuable, the societal relevance of the present project can not only be found in the discussion of highly relevant privacy questions but also in investigating how different sequences can be included into one predictive model. Being able to utilize the power of various independent sequences for prediction is not limited to the domain of app usage but also provides several possibilities for real-world applications in other fields. The most prominent example is the increasing amount of recorded sensor data in industry, health, and also on a personal level. The present research thus provides a first framework which can be utilized as a starting point for addressing prediction tasks related to similar data structures.

7 CONCLUSION

To conclude, the main research question can be answered by looking at the answers for the proposed sub-questions:

RQ1 *To what extent will the use of an LSTM architecture improve the accuracy for dwell time predictions on an application-based level compared to a feed-forward neural network, namely MLP, both trained user-independently?*

The answer to the first research question could be found by testing both an MLP and an LSTM for the task of dwell time predictions. Table 3 shows that the LSTM was able to improve upon the MLPs performance by 0.02, which can be explained by its ability to leverage sequential dependencies for the prediction task.

RQ2 *To what extent will the proposed user-independent LSTM structure bias the results towards users for whom more training data is available?*

The proposed LSTM showed a small bias towards users for whom more training data was available, which can be seen in Figure 11.

RQ3 *Which feature subset can reach the best predictive performance for the task of dwell time prediction for both MLP and LSTM, respectively, is it possible to minimize user-related and contextual features to prevent privacy concerns while keeping the accuracy of the proposed models high, and what differences can be seen between both models using different feature subsets?*

While the LSTM was able to reach a more stable performance when reducing the utilized features, the performance of the MLP dropped rapidly, which can be seen in Table 4. Thus, it can be concluded that an LSTM is more suited for reaching a good trade-off between users' privacy and accuracy.

RQ4 *To what extent can the proposed LSTM structure solve the cold-start problem?*

By looking at Table 5, it can be seen that the LSTM reached a performance of 0.409 on the cold-start problem, performing better than all baselines and the MLP. Accordingly, the proposed LSTM can be used to generalize to previously unseen users although there might be room for improvement by using more complex model structures or approaches like transfer-learning.

8 DATA SOURCE/CODE/ETHICS STATEMENT

Work on this thesis did not involve collecting data from human participants or animals. The main dataset that was used for this project was provided by my thesis supervisor who remains the original owner of the data during and after completion of this thesis. I fully acknowledge that I do not have any legal claim to this data. The dataset used is not publicly available.
An additional dataset was obtained from Kaggle, and is available under the linked repository:
https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps.
The code of this thesis was inspired by various online sources, most importantly by the code published by Jason Brownlee on Machine Learning Mastery (https://machinelearningmastery.com/) and John Wittenauer, published on Medium (https://medium.com/@jdwittenauer).
All figures that are depicted were created by the author of this thesis. When they were inspired by other works, this is explicitly mentioned. The code of this thesis is partly available in the following repository: https://github.com/KatharinaPritzl/MasterThesis

REFERENCES

Aalbers, G., Keijsers, L., Abeele, M. V., & Hendrickson, A. T. (2020). *Preregistration of data collection.* Retrieved from https://osf.io/6fs92/

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Retrieved from https://www.tensorflow.org/

Anysz, H., Zbiciak, A., & Ibadov, N. (2016). The influence of input data standardization method on prediction accuracy of artificial neural networks. *Procedia Engineering*, *153*, 66–70.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., ... others (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion*, *58*, 82–115.

Brownlee, J. (2020a). *Time Series forecasting as supervised learning.* Retrieved from https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

Brownlee, J. (2020b). *Understanding stateful LSTM recurrent neural networks in python with keras.* Retrieved from https://machinelearningmastery.com/understanding-stateful-lstm-recurrent-neural-networks-python-keras/

Cahuantzi, R., Chen, X., & Güttel, S. (2021). A comparison of LSTM and GRU networks for learning symbolic sequences. *CoRR*, *abs/2107.02248*. Retrieved from https://arxiv.org/abs/2107.02248

Cao, H., & Lin, M. (2017). Mining smartphone data for app usage prediction and recommendations: A survey. *Pervasive and Mobile Computing*, *37*, 1–22.

Chen, C., Maekawa, T., Amagata, D., & Hara, T. (2021). Predicting Next-use Mobile Apps Using App Semantic Representations. *Journal of Information Processing*, *29*, 597–609.

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR*, *abs/1409.1259*. Retrieved from http://arxiv.org/abs/1409.1259

Chollet, F., et al. (2015). *Keras.* https://keras.io.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Graves, A. (2013). Generating Sequences with Recurrent Neural Networks. *CoRR*, *abs/1308.0850*. Retrieved from http://arxiv.org/abs/1308.0850

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222-2232. doi: 10.1109/

TNNLS.2016.2582924

Guo, C., & Berkhahn, F. (2016). Entity Embeddings of Categorical Variables. *CoRR*, *abs/1604.06737*. Retrieved from http://arxiv.org/abs/1604.06737

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from https://doi.org/10.1038/s41586-020-2649-2 doi: 10.1038/s41586-020-2649-2

Hill, W. (2017). *Geohashing. What is geohashing exactly?* Medium. Retrieved from https://medium.com/@bkawk/geohashing-20b282fc9655

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Howard, J., & Gugger, S. (2020). *Deep Learning for Coders with Fastai and Pytorch: AI Applications without a PhD*. O'Reilly Media, Incorporated. Retrieved from https://books.google.no/books?id=xd6LxgEACAAJ

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(03), 90–95.

Katsarou, K., Yu, G., & Beierle, F. (2022). Whatsnextapp: LSTM-based Next-App Prediction with App Usage Sequences. *IEEE Access*, *10*. doi: 10.1109/ACCESS.2022.3150874

Keras. (n.d.-a). *Keras Documentation: Embedding layer.* Retrieved from https://keras.io/api/layers/core_layers/embedding/#embedding-class

Keras. (n.d.-b). *Keras Documentation: LSTM Layer.* Retrieved from https://keras.io/api/layers/recurrent_layers/lstm/

Lalmas, M., O'Brien, H., & Yom-Tov, E. (2014). Measuring user engagement. *Synthesis lectures on information concepts, retrieval, and services*, *6*(4), 1–132.

Lee, Y., Cho, S., & Choi, J. (2019). App usage prediction for dual display device via two-phase sequence modeling. *Pervasive and Mobile Computing*, *58*.

Lendave, V. (2021). *LSTM Vs GRU in Recurrent Neural Network: A Comparative Study.* Retrieved from https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/

Mathur, A., Lane, N. D., & Kawsar, F. (2016). Engagement-aware computing: Modelling user engagement from mobile contexts. In *Proceedings of the 2016 acm international joint conference on pervasive and ubiquitous computing* (pp. 622–633).

McGinnis, W. (2015). Pygeohash. *https://pypi.org/project/pygeohash/*.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (p. 56 - 61). doi: 10.25080/Majora

-92bf1922-00a

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Prakash, G. (2021). *Google play store apps.* Retrieved from https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps

Rubinsteyn, A., & Feldman, S. (2016). Fancyimpute. An Imputation Library for Python. *URL https://github. com/iskandr/fancyimpute*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Shanker, M., Hu, M. Y., & Hung, M. S. (1996). Effect of data standardization on neural network training. *Omega*, *24*(4), 385–397.

Shin, C., Hong, J.-H., & Dey, A. K. (2012). Understanding and prediction of mobile application usage for smart phones. In *proceedings of the 2012 acm conference on ubiquitous computing* (pp. 173–182).

Tian, Y., Zhou, K., & Pelleg, D. (2021). What and How Long: Prediction of Mobile App Engagement. *ACM Trans. Inf. Syst.*, *40*(1). Retrieved from https://doi.org/10.1145/3464301 doi: 10.1145/3464301

Vasiloudis, T., Vahabi, H., Kravitz, R., & Rashkov, V. (2017). Predicting session length in media streaming. In *Proceedings of the 40th international acm sigir conference on research and development in information retrieval* (pp. 977–980).

Waskom, M. L. (2021). Seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021.

Wu, S., Rizoiu, M.-A., & Xie, L. (2018). Beyond views: Measuring and predicting engagement in online videos. In *Twelfth international aaai conference on web and social media.*

Xu, S., Li, W., Zhang, X., Gao, S., Zhan, T., & Lu, S. (2020). Predicting and recommending the next smartphone apps based on recurrent neural network. *CCF Transactions on Pervasive Computing and Interaction*, *2*(4), 314–328.

Yang, S., Yu, X., & Zhou, Y. (2020). LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example. In *2020 international workshop on electronic communication and artificial intelligence (iwecai)* (p. 98-101). doi: 10.1109/IWECAI50956.2020.00027

Zou, X., Zhang, W., Li, S., & Pan, G. (2013). Prophet: What app you wish to use next. In *Proceedings of the 2013 acm conference on pervasive and ubiquitous computing adjunct publication* (pp. 167–170).

A APPENDIX A

Abbreviation	Meaning
FFNN	Feedforward neural network
MLP	Multilayer perceptron
LSTM	Long short-term memory
GRU	Gated recurrent unit
RNN	Recurrent neural network
B	Batch size
N	Number of neurons
L	Lookback range
Emd	Embeddings layer
EDA	Exploratory data analysis

Table 6: Abbreviations which are used throughout the present work and their explanation

B APPENDIX B

Mathematical formulation of an LSTM based on Christopher Olah (2015), where W refers to the different weight matrices, and b to the biases:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4}$$

$$C_t = f_t \cdot C_{h-1} + i_t \cdot \tilde{C}_t \tag{5}$$

$$o_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_o) \tag{6}$$

$$h_t = \sigma \cdot tanh(C_t) \tag{7}$$

C APPENDIX C

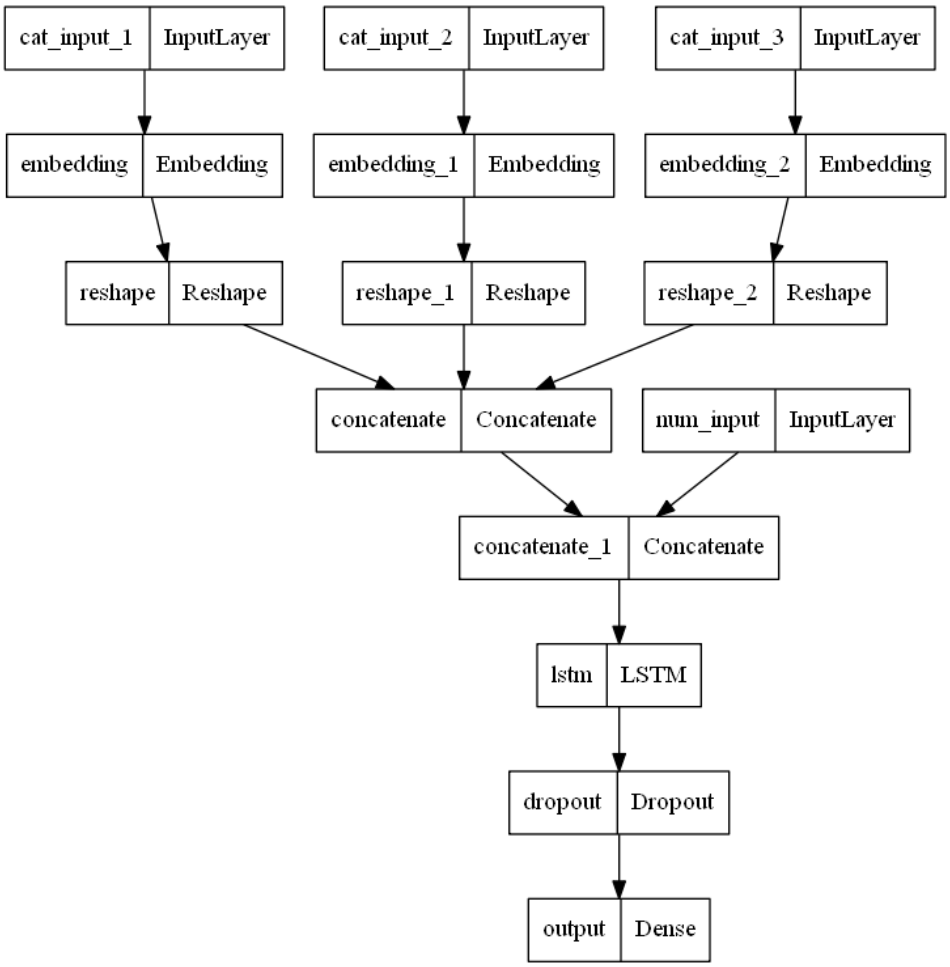

Figure 13: Illustration of the implementation of the proposed LSTM architecture with an embeddings and dropout layer depicted with only three categorical features. Source: the author's illustration obtained in Python by using Keras

D APPENDIX D

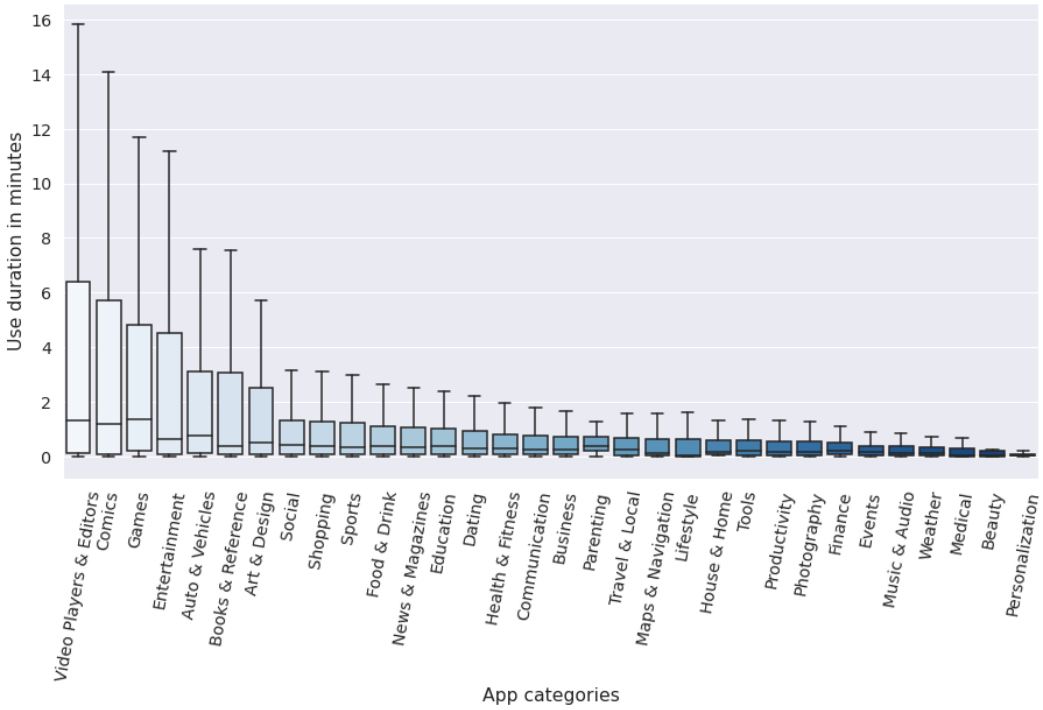

Figure 14: Boxplots of the use duration in minutes per app category. Source: the author's illustration obtained using Python inspired by Tian et al. (2021)

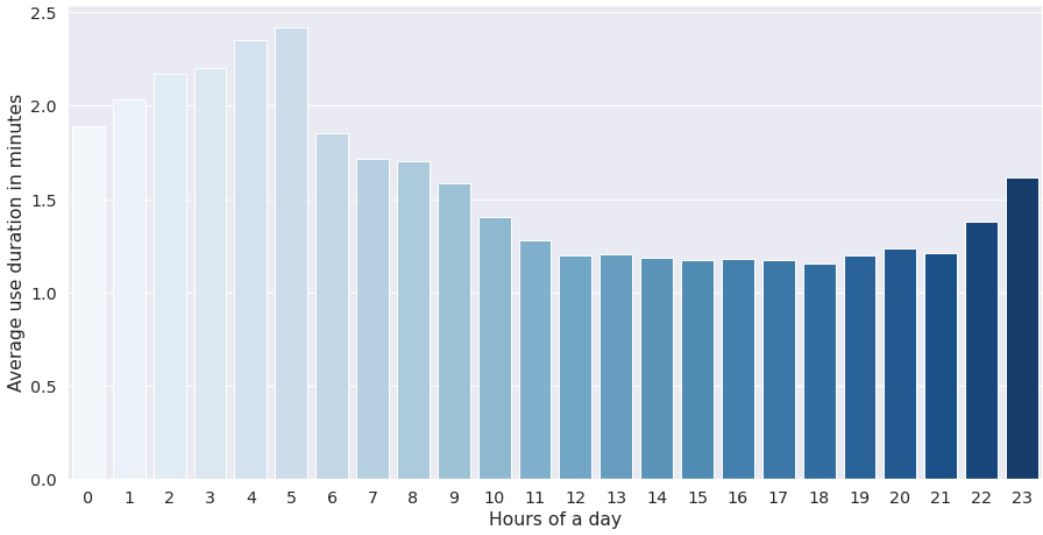

Figure 15: Average app use duration per hour of the day. Source: the author's illustration obtained using Python inspired by Tian et al. (2021)

E APPENDIX E

MLP without Embeddings

full feature subset

Batch Size	Number of Neurons		
	64	128	256
64	0.3883	0.3885	0.3891
128	0.3895	**0.3897**	0.3889
256	0.3890	0.3892	0.3894

Table 7: Scores of the full hyperparameter tuning of the MLP without Embeddings on the full feature subset derived from the validation set

MLP with Embeddings

full feature subset incl. oracle ones

Batch Size	Number of Neurons		
	64	128	256
64	0.4409	0.4436	0.4445
128	0.4425	0.4444	0.4450
256	0.4428	0.4445	**0.4463**

full feature subset

Batch Size	Number of Neurons		
	64	128	256
64	0.4102	0.4105	0.4103
128	0.4106	0.4116	0.4109
256	0.4100	0.4119	**0.4120**

contextual + phone history features

Batch Size	Number of Neurons		
	64	128	256
64	0.4034	0.4035	0.4033
128	0.4039	0.4041	0.4054
256	0.4054	**0.4055**	0.4052

only phone history features

Batch Size	Number of Neurons		
	64	128	256
64	0.3728	0.3730	0.3727
128	0.3727	0.3730	0.3729
256	0.3727	0.3727	**0.3731**

Table 8: Scores of the full hyperparameter tuning of the MLP with Embeddings on different feature subsets derived from the validation set

Cold-start MLP on full feature subset

full feature subset			
	Number of Neurons		
Batch Size	64	128	256
64	0.3749	0.3738	**0.3754**
128	0.3736	0.3730	0.3729
256	0.3712	0.3704	0.3728

Table 9: Scores of the full hyperparameter tuning of the cold-start MLP with embeddings on the full feature subset derived from the validation set

LSTM with lookback range 10

full feature subset incl. oracle ones			
	Number of Neurons		
Batch Size	64	128	256
64	0.4572	0.4602	0.4598
128	0.4579	0.4600	0.4615
256	0.4571	0.4593	**0.4619**

full feature subset			
	Number of Neurons		
Batch Size	64	128	256
64	0.4255	0.4271	0.4263
128	0.4267	0.4264	0.4275
256	0.4267	0.4276	**0.4279**

contextual + phone history features			
	Number of Neurons		
Batch Size	64	128	256
64	0.4246	0.4245	0.4248
128	0.4254	0.4255	0.4254
256	0.4251	**0.4261**	0.4256

only phone history features			
	Number of Neurons		
Batch Size	64	128	256
64	0.4176	0.4165	0.4180
128	0.4174	0.4182	**0.4183**
256	0.4177	0.4178	0.4182

Table 10: Scores of the hyperparameter tuning of the LSTM regarding the number of neurons and the batch size on different feature subset derived from the validation set using the lookback range of 10

Cold-start LSTM with lookback range 10

full feature subset			
	Number of Neurons		
Batch Size	64	128	256
64	0.4054	0.4054	**0.4058**
128	0.4042	0.4045	0.4052
256	0.4027	0.4033	0.4046

Table 11: Scores of the hyperparameter tuning of the cold-start LSTM regarding the number of neurons and the batch size on the full feature subset derived from the validation set using the lookback range of 10

All LSTMs with different lookback ranges

	Accuracy		
	L: 5	L: 10	L:20
full subset incl. oracle	0.4554	0.4619	**0.4636**
full subset	0.4212	0.4279	**0.4296**
contextual and history	0.4185	0.4261	**0.4275**
only history	0.4086	0.4181	**0.4225**
cold-start	0.3970	0.4058	**0.4061**

Table 12: Scores of the hyperparameter tuning of the LSTM regarding the lookback range denoted as L on different feature subsets and for the cold-start problem on the validation set

F APPENDIX F

	train	valid	test	training time
1	0.4181	0.4114	0.4059	0:18:22
2	0.4199	0.4118	0.4060	0:21:32
3	0.4205	0.4129	0.4063	0:22:32
Average	**0.4195**	**0.4120**	**0.4061**	**0:20:48**

Table 13: Performance of MLP trained on the full feature subset over three different training periods with a resampled order of the training records using the best hyperparameters including the training time and the average

	train	valid	test	training time
1	0.4419	0.4295	0.4256	11:07:10
2	0.4369	0.4296	0.4273	9:57:10
3	0.4379	0.4296	0.4257	10:36:00
Average	**0.4389**	**0.4296**	**0.4262**	**10:33:27**

Table 14: Performance of LSTM trained on the full feature subset over three different training periods with a resampled order of the training batches using the best hyperparameters including the training time and the average

G APPENDIX G

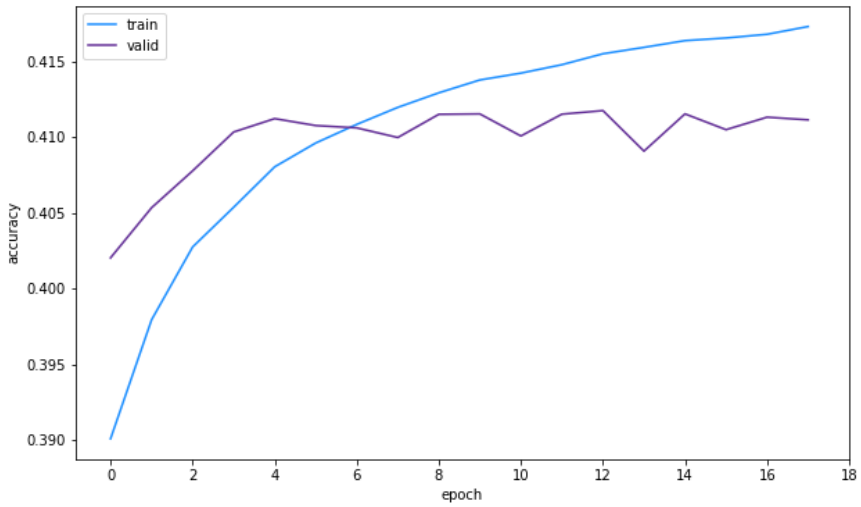

Figure 16: Accuracy plot of the training phase of the MLP on the full feature subset by the use of the best hyperparameters showing training and validation accuracy in comparison. Source: the author's illustration obtained using Python

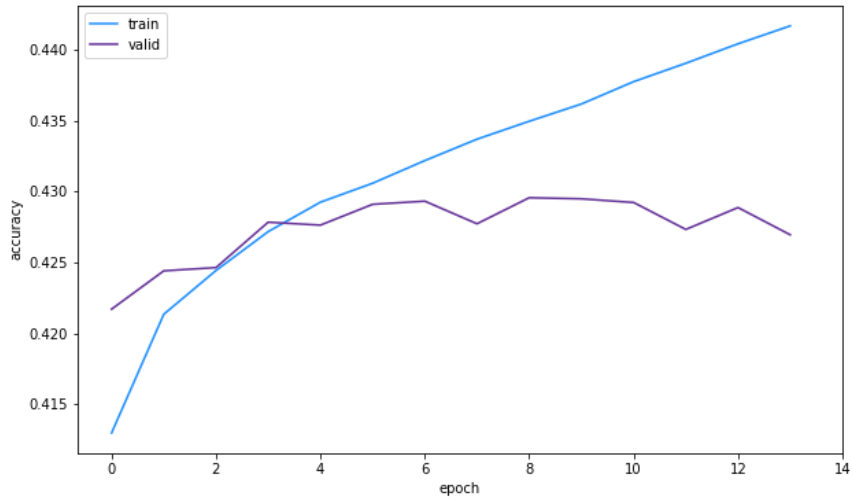

Figure 17: Accuracy plot of the training phase of the LSTM on the full feature subset by the use of the best hyperparameters showing training and validation accuracy in comparison. Source: the author's illustration obtained using Python

H APPENDIX H

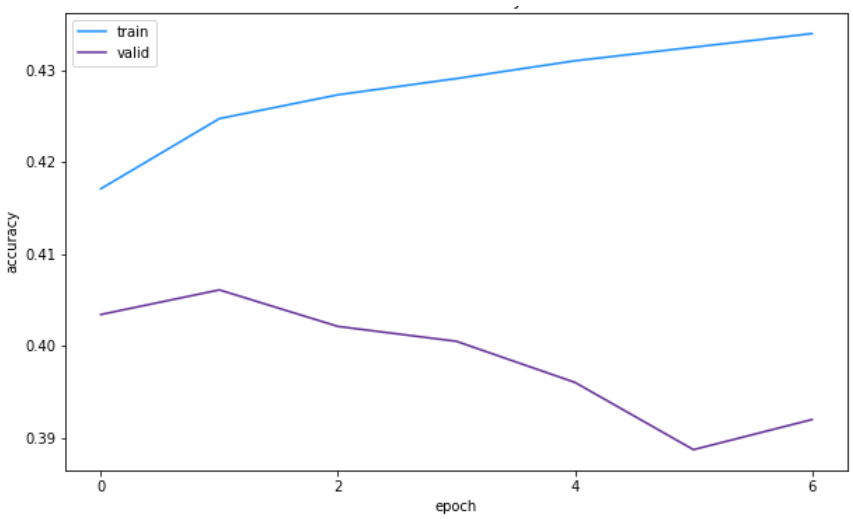

Figure 18: Accuracy plot of the training phase of the LSTM for the cold-start problem, using the full feature subset and the best hyperparameters, showing training and validation accuracy in comparison. Source: the author's illustration obtained using Python

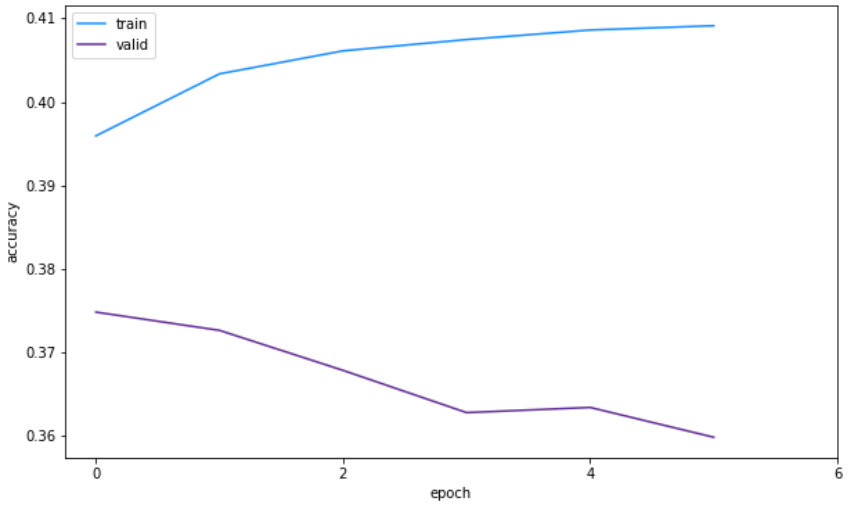

Figure 19: Accuracy plot of the training phase of the MLP for the cold-start problem on the full feature subset, using the best hyperparameters, and showing training and validation accuracy in comparison. Source: the author's illustration obtained using Python