LMU Munich
Frauenlobstraße 7a
D–80337 München
Institut für Informatik

Master Thesis

Intent Prediction with Vectorized Sequential UI Tree Data

August Oberhauser

august.oberhauser@campus.lmu.de

Course of Study: Medieninformatik

Examiner: Prof. Dr. Sven Mayer

Supervisor: Florian Bemmann, M.Sc.

Commenced: June 20, 2023

Completed: January 8, 2024

Abstract

The interaction of a user with an end device such as a smartphone or a computer is very diverse and difficult to predict. Nevertheless, user-specific (personalized) as well as global (collaborative) patterns can possibly be worked out with the help of preceding user interactions. These could be used to predict the intention of a user or a group of users. It is interesting to know to what level of detail these predictions can be made reliably. By making use of the continous on-device data an attempt can be made to gain more insights in the user behavior or even forecast their next actions.

It suggests itself to implement this with the help of user interactions in sessions on Android devices. For this purpose, the Sequential UI Tree data of the device could be tracked, filtered and labeled and then trained with a machine learning model to find similar interaction sequences and then make predictions. These can then be very coarse, such as predicting the next app. Or they can be very detailed, e.g., determining the next user action, such as filling out a form field.

A concept will be developed on how a model for predicting user intent could be built and how it could be applied to the user session. To this end, possibilities for collecting and vectorizing sequential UI trees (e.g., from the Android Accessibility Service) will be discussed (e.g., via Recurrent Neural Network (*RNN*) [18] [2] [7], *Seq2Seq* Model [3], *Screen2Vec* Model [14], *Intention2Text* [21], *Html2Vec* [20]), which are designed to predict the user intent. Here, privacy and feature pre-filtering in UI data plays an important role. After that, personalized as well as collaborative data can be used in a hybrid approach. This model should then be made available to the user in an Android app service and, depending on the level of detail, suggest upcoming apps or actions to the user at a suitable time. It should also be considered whether the user can contribute to the learning process and improve suggested actions through feedback (labeling). The performance of the model can be measured, for example, by indicators such as the amount of training data and time spent on the learning process. The effectiveness can be evaluated by accuracy metrics in predicting, for example, app categories [16] or complete test sequences via Rico [4] or ERICA [5].

Furthermore, the machine learning model could provide the following benefits in addition to intent prediction:

- reduction of the complexity and size of the UI tree
- creation of user groups that have similar behavior when using digital UI systems [10]
- elimination of technical expertise on individual features that would be required to manually compare user sessions [9]
- consideration of a user's history over time (sequential)
- comparison of user interactions without providing privacy invasive information
- supporting app developers to improve their app design and usability
- application in psychology and market research
- pre-loading of processes on devices (energy savings) [19]

As listed above, many fields of application can profit by elaborating such a system. It would be exciting to know, how the concrete concept would look like and if it can be implemented successfully e.g. to improve the user experience on end-user devices.

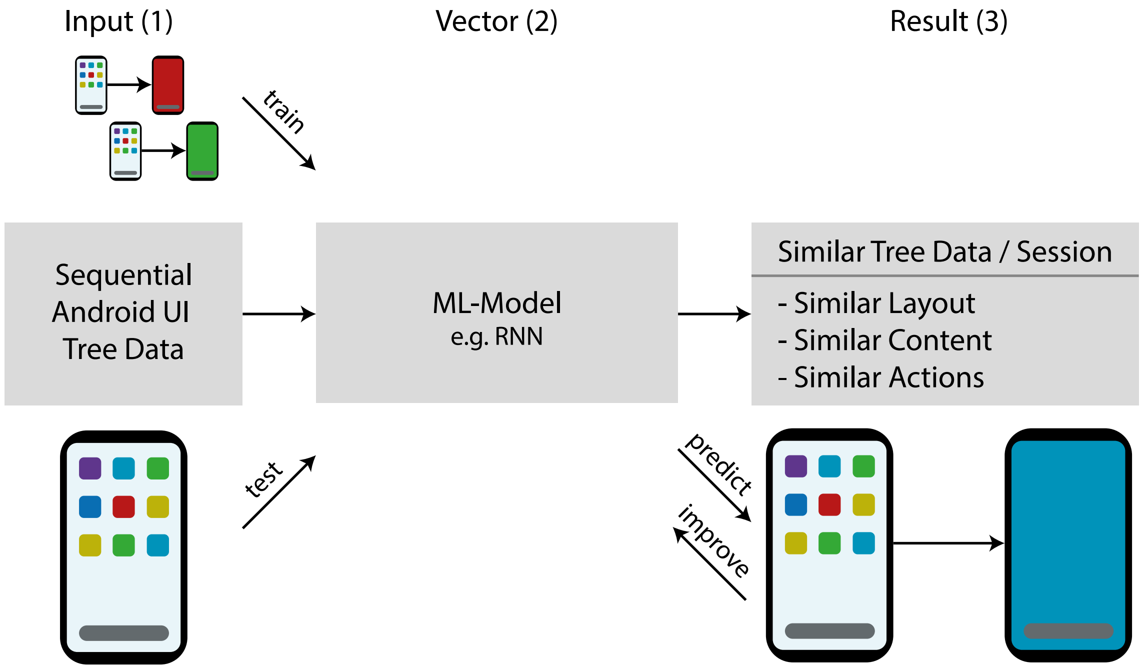


Figure 1: Possible procedure using a Machine-Learning algorithm to predict the next intent from a beginning user session: The input (1) can be a sequence of Android tree data. With help of a Machine-Learning-Model (2) (e.g. RNN) a vector representation can be trained and then predict the most probable action or screen (3) from a given starting sequence, but also can be improvde through the users feedback.

Contents

1 Introduction **15**
1.1 Necessarity of Vectors for Android UI 15

2 Related Work **17**
2.1 Datasets of UI Trees 17
2.1.1 ERICA 17
2.1.2 Rico / RicoSCA 18
2.1.3 Mobile UI CLAY Dataset 18
2.2 Vector models 18
2.2.1 Doc2Vec and Word2Vec 20
2.2.2 Screen2Vec 20
2.2.3 Screen2Words 20
2.2.4 Intention2Text 20
2.2.5 Html2Vec 20
2.2.6 Tree2Vec 21
2.2.7 Activity2Vec 21
2.3 Time Series / Sequence models 21
2.3.1 Seq2Seq Model 21
2.3.2 Click Sequence Prediction / PathFinder 21
2.3.3 Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning 21

3 Theoretical Framework **23**
3.1 Android UI Data 23
3.1.1 Data tree structure 23
3.1.2 Retrieval of UI data via Android Accessibility Service 23
3.2 Machine Learning 24
3.2.1 Preprocessing 25
3.2.1.1 Feature selection 25
3.2.1.2 Missing data 25
3.2.1.3 Normalization and Standardization 25
3.2.1.4 Padding 26
3.2.1.5 Categorical Variables 26
3.2.2 Supervised vs Unsupervised vs Semisupervised 26
3.2.2.1 Supervised Learning 26
3.2.2.2 Unsupervised Learning 26
3.2.3 Under and Overfitting 27
3.2.4 Evaluation Metrics 27

3.3 Artificial Neural Nets 27
3.3.1 Classes of Neural Nets 27
3.3.1.1 Deep Neural Nets 27
3.3.1.2 Convolutional Neural Nets 27
3.3.1.3 Recurrent Neural Networks and LSTMs / GRU 27
3.3.1.4 Transformer 27
3.3.1.5 Autoencoders 28
3.3.2 Tensorflow and Keras 28
3.4 Evaluation and Metrics 28
3.4.1 Mean Squared Error 28
3.4.2 F1 Score 28

4 Methodology **29**
4.1 Data Aquisition 29
4.2 Methodological variety 29
4.3 Methodological choices 30
4.4 Research Biases 30

5 Results **31**
5.1 Datasets 31
5.1.1 Rico 31
5.2 Preprocessing Android UI tree data 31
5.2.1 Filtering privacy invasive details 31
5.2.2 Normalization, Feature selection 31
5.3 Model 31
5.4 Evaluation 32
5.4.1 Mean Squared Error 32
5.4.2 F1 Score 32
5.5 Limitations 32

6 Application of Android UI tree vectors **33**
6.1 Automation and testing of Android apps 33
6.2 UI design similarities 33
6.3 Action prediction models, User behavior modeling 33
6.4 Behavioral analyses for smartphone usage patterns 33

7 Conclusion and Future Work **35**

List of Figures

1 Possible procedure using a Machine-Learning algorithm to predict the next intent from a beginning user session: The input (1) can be a sequence of Android tree data. With help of a Machine-Learning-Model (2) (e.g. RNN) a vector representation can be trained and then predict the most probable action or screen (3) from a given starting sequence, but also can be improvde through the users feedback. 3

2.1 View element insights [22] 22

3.1 https://developer.android.com/studio/debug/layout-inspector, https://github.com/mimuc/app-ins-gruene 23

5.1 Model loss vs validation loss 32

List of Tables

2.1 Attributes of a view hierarchy record 19

Glossary

accuracy

[TODO]: Accuracy explained in Udemy

. 20, 22

Big Data Extremely large and complex data sets which can only be processed with modern computing soft- and hardware. 24

rooting Rooting is a method to gain privileged access to the operating system Android. 21

Acronyms

Continuous Bag-of-Words (CBOW) . 20

Convolutional Neural Network (CNN) . 18

Graphical User Interface (GUI) . 20

Graph Neural Network (GNN) . 18

Long Short-Term Memory (LSTM) . 21, 22

Machine Learning (ML) Scientific approach to form statistical models without the need to explicitly program it. 24, 25

Neural Network (NN) . 20

Operating System (OS) . 21

Recurrent Neural Network (RNN) . 20

Residual Neural Network (ResNet) . 18

Social Networking Service (SNS) . 21

1 Introduction

This is a typical human-computer interaction thesis structure for an introduction which is structured in four paragraphs as follows:

1.1 Necessarity of Vectors for Android UI

Motivation for transforming Android UI tree data to vectors

- Low Button depth: number of clicks until one gets to the action [12]

Contributions: [22] • An analysis of a large-scale dataset of mobile user click se- quences that reveals rich factors and complexity in modeling click behaviors, which contributes new knowledge to under- stand mobile interaction behaviors. • A Transformer-based deep model that predicts next element to click based on the user click history and the current screen and time. The model does not rely on a vocabulary of prede- fined UI elements and provides a general solution for modeling arbitrary UI elements for click prediction. • A thorough experiment that compares our deep model with multiple alternative designs and baseline models, and an analysis of model behaviors and benefits that the model can bring to improve mobile interaction.

Contributions: [**screen2vec**] Screen2Vec: a new self-supervised technique for generating more comprehensive semantic embeddings of GUI screens and components using their textual content, visual design and layout patterns, and app meta-data. (2) An open-sourced GUI embedding model trained using the Screen2Vec technique on the Rico [9] dataset that can be used off-the-shelf. (3) Several sample downstream tasks that showcase the model's usefulness.

[TODO]: P1.1. What is the large scope of the problem?

[TODO]: P1.2. What is the specific problem?

[TODO]: P2.1. The second paragraph should be about what have others been doing

[TODO]: P2.2. Why is the problem important? Why was this work carried out?

[TODO]: P3.1. What have you done?

[TODO]: P3.2. What is new about your work?

[TODO]: P4.1. What did you find out? What are the concrete results?

[TODO]: P4.2. What are the implications? What does this mean for the bigger picture?

2 Related Work

[TODO]: introduce in related work, differentiate the importance of datasets

Describe relevant scientific literature related to your work.

2.1 Datasets of UI Trees

Requirements on a good data set - Google and Samsung - need to have correct data - needs enough data to train a NN - need enough features to be able to recognize patterns - up to date - Publicly available - Variable length of app sessions, define one session of activating the screen until it is turned off.

Missing - System to feed in in real time - Dataset which is across multiple apps, also tracks the system -

2.1.1 ERICA

ERICA is a design and interaction mining application, which allows gathering *interaction traces* by capturing the users activity on Android apps [5]. This is accomplished through a web-based interaction layer in contrast to the other common approach of using *accessibility services* directly. They justify that approach by the lack of need to install additional applications, as only a browser is required. A further reason is the response latency of the commonly used *UiAutomator*, which cannot collect the data in time. Also they argue that capturing and simultaneously interacting with the apps may overload the user device and challenges the user experience. Therefore the much more powerful servers take the task of capturing the UI trees. The apps are hosted on multiple physical devices with a modified Android OS directly connected with the server. ERICA captures UI screens and user flows by tracking UI changes. They then used this data to form k-mean clusters from the UI elements (visual and textual features) and the interactive elements (icons and buttons). Based on the clusters they then build classifiers and trained an AutoEncoder (3.3.1.5) to determine the flows from the test dataset. The authors worked out 23 common user flows (from over a thousand popular Android apps) which aim to provide complementary, promising or new design patterns and trends.

2.1.2 Rico / RicoSCA

Rico [4] (spanish for "rich") is the successor of ERICA. It aims to help perform better at designing and support the creation of adaptive UIs. As far as known to date this is the largest collection of mobile app designs and traces with covering 72k UI screens in 9.7k Android apps. Like its predecessor Rico uses a web-based approach to collect user traces. It enables the applications like searching for designs, generation of UI layouts and code, modeling of user interactions, and prediction of user perception. It exposes visual, textual, structural, and interactive design properties of more than 72k unique UI screens. Unfortunately the dataset doesn't include interaction traces for app to app transitions or interactions with the Android OS itself. In table 2.1 a collection of all view hierarchy attributes is shown with their meaning. These were extracted by iterating over all view hierarchy files contained in the traces of the dataset. This gives insights in what attributes were recorded in the Rico dataset and what relevance they may have during training the model. The authors of Rico used their dataset to train a 64-dimensional UI layout vector **??** with an AutoEncoder 3.3.1.5. For their input they converted the UI layout hierarchy to an image with colored bounding boxes differentiating images and text. This has the advantage to be able to deal with the high dimensions inside the UI tree. But the conversion also most likely discards lots of meaningful information hidden in the UI tree semantics.

The RicoSCA dataset has been formed out of the research topic of mapping language instructions to mobile UI action sequences [15]. They removed screens whose bounding boxes in the view hierarchies are inconsistent with the screenshots with the help of annotators. The process of filtering reduced the Rico dataset to 25k more concise and meaningful screens.

2.1.3 Mobile UI CLAY Dataset

The Google researchers Gang Li et al. [13] present a a so-called *CLAY* pipeline which is able to denoise mobile UI layouts from incorrect nodes or adding further semantics to it. As basis they used the Rico **??** dataset for a subject of improvement. They state that recording results are dynamic and can get out of sync with the actual screen of the user. That leads to 37.4% of screens which contain invalid objects. This induces invisible or misaligned objects, or objects which are not clickable (greyed out). The researchers filtered invalid objects by training a Residual Neural Network (ResNet) model with the screenshots to classify nodes as as invalid if their bounding boxes don't match. Also they introduced two models: a Graph Neural Network (GNN) and a Transformer model to each determine the view type (also related to the view class). For that they considered the view hierarchy attributes as well as the screenshots via a Convolutional Neural Network (CNN). They claim they outperform heuristic approaches for detecting layout objects without a visual valid counterpart and also can recognize their types in more than 85%. This pipeline could help to improve intent prediction algorithms as less inconsistent data is applied to the model.

2.2 Vector models

- Compress a huge data set to a concise model - Vector Representation enables

Advantages: - "small" or smaller than the data set itself - No need to have pre knowledge about the topic, just need input an output (labels) for unsupervised NN -

Key	Type	Shape	Description
Per View			
activity_name	string	(1)	Name of the activity: e.g. "com.my_app.AppName.MainActivity"
is_keyboard_deployed	bool	(1)	Indicates if the keyboard is shown
request_id	int	(1)	Id used by the crawler to request the view
Per Node			
abs-pos	bool	(1)	Indicates if position in *bounds* is relative or absolute; if *true*, *rel-bounds* is set
adapter-view	bool	(1)	Indicates that children are loaded via an adapter, see [6]
ancestors	[string]	(None)	Ancestors of current node, e.g. "android.view.View"
bounds	[integer]	(4)	Absolute or relative boundaries, dependent on *abs-pos*
children	[node]	(None)	Child nodes
class	string	(1)	"com.my_app.lib.ui.views.DropDownSpinner"
clickable	bool	(1)	User can interact by press / click
content-desc	string	(1)	(Accessibility) description of the node "Interstitial close button"
draw	bool	(1)	Indicates if this node is drawn on the canvas
enabled	bool	(1)	Indicates if this node is in the enabled state
focusable	bool	(1)	Indicates if this node can be focused
focused	bool	(1)	Indicates if this node can is currently in focus
font-family	string	(1)	States the font family, e.g. "sans-serif"
long-clickable	bool	(1)	Indicates if this node has a long press action
package	string	(1)	States which packages the node belongs to "com.my_app.mypackage"
pressed	bool	(1)	Indicates if this node can is currently pressed
rel-bounds	[integer]	(4)	Relative boundaries, if *abs-pos* is set to *true*
resource-id	string	(1)	The unique resource identifier for this view "android:id/navigationBarBackground"
scrollable-horizontal	bool	(1)	Indicates if this node can be scrolled horizontally
scrollable-vertical	bool	(1)	Indicates if this node can be scrolled vertically
selected	bool	(1)	Indicates if this node can is currently selected
text	string	(1)	Text value if this node is a textual element
text-hint	bool	(1)	Explanation text for text boxes or icons
visibility	string	(1)	Indicates if this node is hidden, e.g. "visible", "gone"
visible-to-user	bool	(1)	Indicates if this node can be seen in the viewport by the user

Table 2.1: Collection of attributes of a *view hierarchy* record, extracted from all interaction traces of the Rico [4] dataset.

2.2.1 Doc2Vec and Word2Vec

[11]

2.2.2 Screen2Vec

Toby Jia-Jun Li, Lindsay Popowski et al. [14] wrote a Neural Network (NN) called Screen2Vec which embeds the UI components while preserving the semantics. It is claimed that they are among the first to develop a NN for mobile screens which takes textual, visual design, and layout patterns and app context meta-data into account. As inspiration they used the Word2Vec **??** to predict result by considering the context and map them to a Continuous Bag-of-Words (CBOW). The self-supervised **??** model consists of two pipeline levels. The outer level (Graphical User Interface (GUI) screen level) combines embeddings of GUI components, layout hierarchy and app descriptions. The inner level is only present for the GUI components as they contain nested embeddings for the screen text and the class type. The screen text (in the inner level) as well as the app description (in the outer level) is processed using a pretrained Sentence-BERT model. The layout hierarchy is converted to a colored image encoding the text and image boundaries with colors (like in 2.1.2). With such model, a vector can be calculated for each screen which then can be compared to each other, e.g. by the euclidean distance between the pixel representations, or comparing the distance in the view hierarchy representation. When taking all features into account, both, the euclidean and the hierarchical approach, get an accuracy of around 0.85 that the correct screen is among the first 1% of the models predictions. In around half of the cases the predicted screen ("Top-1 Accuracy") is the correct one.

Such an approach of representing an Android layout and context in a vector can be used as pretrained embedding for feeding a Recurrent Neural Network (RNN) predicting upcoming screens.

2.2.3 Screen2Words

2.2.4 Intention2Text

[21]

2.2.5 Html2Vec

[20]

2.2.6 Tree2Vec

2.2.7 Activity2Vec

2.3 Time Series / Sequence models

- one more dimension - allows predicting unseen states - back propagation -> see technical part -

RNNs: 9_Personalizing session based recommendations with RNNs [18] 10_Bansal_Hybrid RNN Recommender system [2] [7]

2.3.1 Seq2Seq Model

[3]

2.3.2 Click Sequence Prediction / PathFinder

Seokjun Lee et al. [12] propose a technique called *PathFinder* which aims to predict the sequence of user clicks in Android mobile apps. The user input and the contextual data is collected via the Android Accessibility Services **??**, so the users Operating System (OS) does not need to be modified or rooted. They collected the data from 55 students of their university with a sequence tracing tool and collected near 2 million button clicks from over a thousand apps. They follow a collaborative and content-based approach which takes both all the users data as well as the individual preferences into account. The *button depth* describes the number of clicks or taps until the user gets to their target screen. In average nearly the user has 16 buttons as candidates to press as the next action. With a personalized UI a the *button depth* should decrease significantly. The next user click is dependent on very recent but also on previous clicks happened a longer time ago, e.g. taking a picture relates to uploading it later to their Social Networking Service (SNS). The authors train a Long Short-Term Memory (LSTM) model to predict the next button, which will be clicked on. PathFinder predicts the most probable three buttons with a 0.76 F-measure.

In contrast to this work, *PathFinder* does not take into account the complete view hierarchy or other spatiotemporal information. Just the previous and the current app and the click history with their button properties are considered. Also as far as known the dataset and code is not publicly accessible.

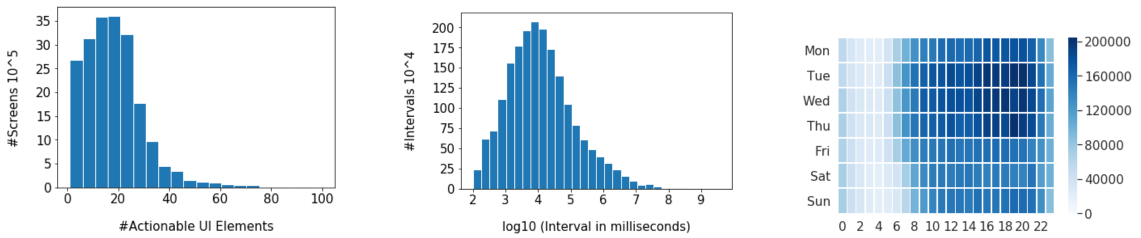
2.3.3 Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning

Xin Zhou and Yang Li [22] extend the work of [12]. They expect to optimize the UI experience by recommending the users next click interaction based on their findings. They gathered a dataset of 20 million clicks from 4k mobile users. The goal was to overcome the challenge of accurate but also scalable click sequence modeling. That means that apps are not limited in their composition and the screens get increasingly diverse and there's no predefined set of UI elements. Also the users click behavior is very individual and heavily dependents on situational factors.

Based on the Transformer architecture in 3.3.1.4 they created a deep learning model which has 48% accuracy for predicting the next user click and 71% accuracy for the three most probable *actionable* objects. The researchers differentiated three main inputs for embedding their elements visible on the screen: the text content, the type of view and the bounding box. All the elements are then passed to a Transformer Encoder representing a single screen. Together with the click event as well as the time encoding the screen embedding serves as one concatenated input step. The encoded time can be very different as it doesn't follow a regular sample interval, but is recorded as soon as the screen is changing 2.1 (b). Multiple input steps then form a sequence fed in a second Transformer Encoder which contains all past screens and clicks. The current screen embedding and time are then passed to a pointer (M-layer perceptron) which calculates the most probable *actionable* elements to be clicked on. They consider a UI element as *actionable*, if it is currently `clickable`, `visible` and `enabled` (cf. 2.1). More complex screens can have much more actionable elements, which makes the prediction much more difficult 2.1 (a).

This paper solves a lot of problems previous attempts had. The dataset includes cross-app transitions which make 26% of all clicks, which are also considered in their model. The current context was taken into account such as the time of the day and day in the week 2.1 (c) which adds a lot of semantics. Further they used a transformer model with self-attention, which reduces the training times significantly compared to LSTM. Compared to the approach tested in this paper, they also predict the concrete element that will be clicked on instead of the absolute screen coordinates.

Unfortunately as far as the investigation permits, neither the dataset nor the code were publicly provided.



(a) Number of actionable elements per screen

(b) The distribution of time intervals between click events

(c) The distribution of time intervals between click events

Figure 2.1: View element insights [22]

3 Theoretical Framework

3.1 Android UI Data

3.1.1 Data tree structure

Mean 18 actionable elements, with Std=12. [22]

3.1.2 Retrieval of UI data via Android Accessibility Service

Semantics tree: https://developer.android.com/jetpack/compose/semantics https://android.googlesource.com/platform/fr
dev/uiautomator/library/src/com/android/uiautomator/core/AccessibilityNodeInfoDumper.java
https://github.com/Gustl22/android-accessibility/blob/c158808533d6fc017455184a7317555d3e6946f6/GlobalActionBa

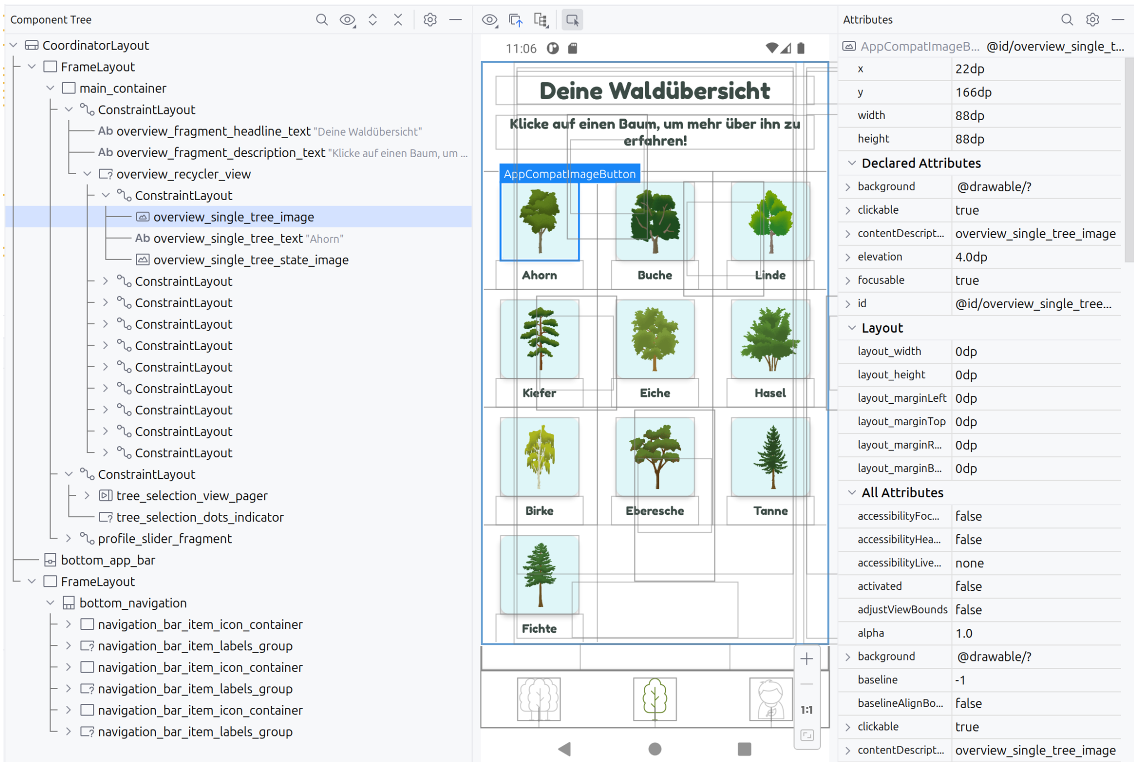


Figure 3.1: https://developer.android.com/studio/debug/layout-inspector,
https://github.com/mimuc/app-ins-gruene

```xml
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<hierarchy>
    <node index="0" text="" resource-id="" class="android.view.ViewGroup"
          package="de.lmu.treeapp" content-desc="" checkable="false"
          checked="false" clickable="false" enabled="true" focusable="false"
          focused="false" scrollable="false" long-clickable="false"
          password="false" selected="false" visible-to-user="true"
          bounds="[0,137][1440,2923]">
        <node index="0" text="" resource-id=""
              class="androidx.viewpager.widget.ViewPager" package="de.lmu.treeapp"
              content-desc="" checkable="false" checked="false" clickable="false"
              enabled="true" focusable="true" focused="false" scrollable="true"
              long-clickable="false" password="false" selected="false"
              visible-to-user="true" bounds="[4,141][1436,2707]">
            <node index="22" text="Eiche" resource-id=""
                  class="android.widget.TextView" package="de.lmu.treeapp"
                  content-desc="" checkable="false" checked="false"
                  clickable="false" enabled="true" focusable="false"
                  focused="false" scrollable="false" long-clickable="false"
                  password="false" selected="false" visible-to-user="true"
                  bounds="[116,182][1324,315]" />
        </node>
        <node index="1" text="" resource-id="" class="android.view.ViewGroup" ...>
            <node NAF="true" index="0" text="" resource-id=""
                  class="android.widget.FrameLayout" package="de.lmu.treeapp" ... />
        </node>
    </node>
</hierarchy>
```

Listing 3.1: Android Accessibility Node in XML.

3.2 Machine Learning

Machine Learning (ML), a term spread by Arthur Lee Samuel, is a method of data analysis, more precisely a scientific approach to form statistical models without the need to explicitly program it [17]. It uses algorithms to iteratively learn how data is structured. In contrast to statistical inference or manually crafted statistical models respectively, ML can solve tasks by automation of model building. Its advantages lie in finding hidden relations and patterns from the context, without having any or only a small pre knowledge of the data, thus it is a strong tool for generalization or abstraction of large datasets, also known as Big Data. ML can be applied to the following fields among others: email and spam filtering, fraud detection, cybersecurity, web search engines, recommender systems (like known from Netflix or Amazon), advertising, translators and text generation, pattern and image recognition. The data driven approach also comes with some drawbacks: the outcome heavily depends on the provided data. It can include biases and therefore may acquire forms of discrimination or unfair treatment. Nonetheless ML has a lot of potential to uncover hidden connections in large datasets.

3.2.1 Preprocessing

Preprocessing describes the step after one acquired their data, but before training the ML model. This step is not to be underestimated. A ML model can perform significantly better when certain preprocessing steps are applied [1].

To be able to preprocess the data, we have to know with what kind of data we handle with. Data can occur in different forms, but we can break them down in three main types:

- **Categorical values**: a value is always assigned to a class with fixed pool of predetermined classes. E.g. letters, words, brands, animals, chemical elements
- **Continuous values**: the value can be fractional and may lies in between a lower and an upper bound. E.g. temperature, velocity, geographic position
- **Integer values**: the value is a whole number and may also lie in between a lower and an upper bound. E.g. revolutions per minute, product number, annual sales

For discrete and continuous values, we have a wide variety of options to prepare them to be able to be processed by a ML model [8].

[TODO]: Tensors, Datasets

3.2.1.1 Feature selection

Such as Filtering privacy invasive details

Parameterizing the vectorization process a) Vector length b) Weighting of features c) Manipulating individual parameters of model

3.2.1.2 Missing data

Some data entries may are missing. Therefore, you have two approaches to get around these missing values. One can drop these values by removing the column or row. This is only recommended if you are not relying on this data entry, or this the whole feature is not expected to be important enough to bring any value to the model's performance. Further you can fill the data with a default value like zero or calculate a reasonable value from the surrounding data entries by taking their "mean, median, or interpolation" [8]. The second approach can only be applied to numerical data.

3.2.1.3 Normalization and Standardization

This is only applicable for numerical data. Many ML models work better or exclusively with normalized data. This means that the values have to be in a certain range, most commonly are from **0** to **1** or from **-1** to **1**. This can be achieved by dividing all values with the difference of the minimum and maximum value and shift the output accordingly [8].

Sometimes this is not enough, e.g. if having a few extreme values, and an approach is desired which better reflects the average data. Here the standardization, also called z-score normalization, comes into play. This method scales the values so that the mean value is placed at **0** and the standard deviation is placed at **1**.

3.2.1.4 Padding

3.2.1.5 Categorical Variables

[TODO]: categorical

According to [1] these steps can be removal of emoticons, elimination of stopwords and stemming for text based models.

Category Embedding before LSTM

- Embedding layer Dimension near the actual average length of features (?)

3.2.2 Supervised vs Unsupervised vs Semisupervised

3.2.2.1 Supervised Learning

Supervised: Classification and regression Uses **labeled** examples: Input and output is known

Learns by comparison of the output it is provided with the output the model *predicts*.

Steps: - Data acquisition - Data cleaning / Preprocessing (Panadas) - Split into Training Data, Validation data, and Test data (cannot adapt the model after using the test data) - Train the model with the train data - Evaluate the model with the test data, then can adapt the model by the developer - Last deploy the model to production

3.2.2.2 Unsupervised Learning

Clustering Reinforcement learning

self-supervised

3.2.3 Under and Overfitting

3.2.4 Evaluation Metrics

3.3 Artificial Neural Nets

- Uses biological neuron systems as paradigm to generate mathematical models - can solve tasks by abstraction or generalization of data relations

Activation Functions Cost function Gradient - Regression: Continous Values - Classification: Multiple class - One Class

3.3.1 Classes of Neural Nets

3.3.1.1 Deep Neural Nets

Neural Net with more than one layer - Dense Layer

3.3.1.2 Convolutional Neural Nets

3.3.1.3 Recurrent Neural Networks and LSTMs / GRU

LSTM 4 dimensional

Limitations to only 3 dimensions, needs flattening

Sample dimension (X -> y) Time (Step) Dimension Feature Dimension Data, Quantity dimension, such as Image dimensions, or multiple nodes

TimeDistributedLayer

3.3.1.4 Transformer

[TODO]: Check

The neural net transformer model is a type of deep learning architecture that uses attention mechanisms to process sequential data, such as natural language or speech. It does not rely on recurrent or convolutional layers, which are commonly used in other neural network models. Instead, it uses a combination of self-attention, multi-head attention, and feed-forward layers to encode and decode the input and output sequences12

The difference between encoder and decoder transformer is that they have different roles and sublayers in the model. The encoder transformer takes an input sequence, such as a sentence in one language, and converts it into a vector representation, called an encoding, that captures the meaning and context of the input. The encoder transformer consists of multiple identical layers, each with two sublayers: a multi-head self-attention layer and a feed-forward layer. The self-attention layer allows the encoder to learn the relationships and dependencies between the words in the input sequence. The feed-forward layer applies a non-linear transformation to the output of the self-attention layer

The decoder transformer takes the encoding from the encoder and generates an output sequence, such as a sentence in another language. The decoder transformer also consists of multiple identical layers, each with three sublayers: a masked multi-head self-attention layer, a cross-attention layer, and a feed-forward layer. The masked self-attention layer allows the decoder to learn the relationships and dependencies between the words in the output sequence, but prevents it from attending to the future words that have not been generated yet. The cross-attention layer allows the decoder to attend to the encoding from the encoder and learn the alignment between the input and output sequences. The feed-forward layer applies a non-linear transformation to the output of the cross-attention layer

3.3.1.5 Autoencoders

Encoder, Decoder

3.3.2 Tensorflow and Keras

Layers FlattenLayer

Positive Integer to Dense Vectors of fixed size

3.4 Evaluation and Metrics

3.4.1 Mean Squared Error

3.4.2 F1 Score

4 Methodology

[TODO]: Describe methodology

Start with your overall approach to the research. What research problem or question did you investigate? What type of data did you need to answer it? Quantitative, qualitative, or mixed? Primary or secondary? Experimental or descriptive?

4.1 Data Aquisition

How you collected and analyzed your data Describe the specific methods you used for data collection and analysis. How did you collect and analyze your data? What tools or materials did you use? How did you ensure the quality and accuracy of your data?

- Why RICO, rather use a dataset with accessibility service. - ERICA employs a human-powered approach over an automated one: - More realistic results - required user input, which cannot emulated, Google Captcha, real data - humans detect the completion of UI updates [5] - erica is quite outdated

Any tools or materials you used in the research. The type of research you conducted

E.g. Google Scholar, Google Research, Tensorflow, Keras, Udemy, Open Source, Reproducible

4.2 Methodological variety

Explain why you chose these methods over others. How do they relate to your research question and literature review? How do they address the limitations or gaps in existing research? How do they suit your research design and objectives?

- No similar approach - No dataset present with consecutive sequential app usages - Many different approaches to solve this problem: - Encoder, Decoder, etc... - A Study can follow

4.3 Methodological choices

Evaluate and justify your methodological choices. Why you chose these methods How did they affect the outcome of your research? What challenges or difficulties did you encounter and how did you overcome them? How can you ensure the credibility and generalizability of your findings?

[TODO]: overall goal is more than just interaction traces

Goal: Make the algorithm as independent of the data as possible. Find general rules to feed the data. Applicable to other research fields, not just UI traces. Use LSTM, so predict something unseen, in contrast to RICO or ERICA, which only categorize the current context

4.4 Research Biases

How you mitigated or avoided research biases

How this thesis is working? Apparatus, Procedure, Utilities

5 Results

5.1 Datasets

- Problem with sequential data sets

5.1.1 Rico

- Too less frames.
- No transition between apps.

5.2 Preprocessing Android UI tree data

5.2.1 Filtering privacy invasive details

5.2.2 Normalization, Feature selection

Dealing with variable length data tf.io.VarLenFeature()

5.3 Model

Multiple approaches

AutoEncoder:

- Encoder -> Decoder -> LSTM -> Decoder
- Encoder -> LSTM -> Decoder
- LSTM -> Encoder -> Decoder (AutoEncoder)

Decoder can either only decode to x and y or to whole UI tree.

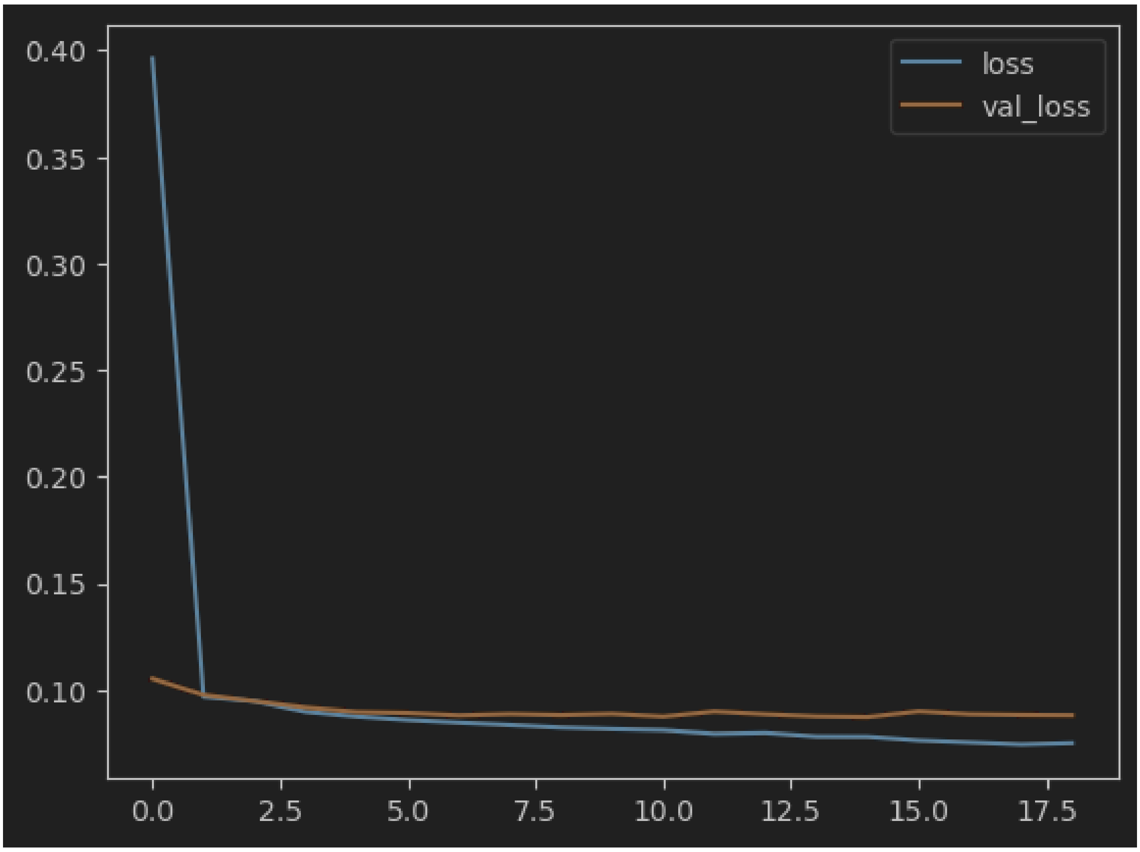


Figure 5.1: Model loss vs validation loss

5.4 Evaluation

5.4.1 Mean Squared Error

5.4.2 F1 Score

5.5 Limitations

Dataset Dataset is not through different apps, only in one app. Dataset is not detailed enough in the time steps, or not containing all data Dataset is not long enough Dataset has no paid apps or apps with login, which most services require Dataset has wrong data see [13]

Preprocessing Need more time to validate what are the core parameters to predict the next user intent

Model needs more investigation on what data is needed How many neurons are required to achieve this Play around with different layers, also Convolutional and pretrained embeddings

6 Application of Android UI tree vectors

6.1 Automation and testing of Android apps

6.2 UI design similarities

6.3 Action prediction models, User behavior modeling

6.4 Behavioral analyses for smartphone usage patterns

7 Conclusion and Future Work

Summary

Outlook

Future directions for research in this area

ChatGPT – Image Recognition – Limitation als Ausblick

Generate Dataset which overcomes the limitations

Make a study with actual feedback on a prediction system, visualization -> Learn faster and directly, Reinforced learning

Work out user flows, like in ERICA, but without the need to separate it from the interaction tree

Take in visual and textual context (semantics).

Make dataset with app overlapping traces. Use dataset with preprocessing such as RicoSCA or Clay. Also use accessibility service, as phones are much more powerful. No need for web interface. -> Reinforced directly on the phone, more privacy.

[TODO]: check if references are still on their own page

References

[1] Saqib Alam, Nianmin Yao. “The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis”. In: *Computational and Mathematical Organization Theory* 25 (2019), pp. 319–335.

[2] Saumya Bansal, Niyati Baliyan. “Remembering past and predicting future: a hybrid recurrent neural network based recommender system”. In: *Journal of Ambient Intelligence and Humanized Computing* (2022), pp. 1–12.

[3] François Chollet. *A ten-minute introduction to sequence-to-sequence learning in Keras*. 2017. URL: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (visited on 02/15/2023).

[4] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, Ranjitha Kumar. “Rico: A mobile app dataset for building data-driven design applications”. In: *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 2017, pp. 845–854.

[5] Biplab Deka, Zifeng Huang, Ranjitha Kumar. “ERICA: Interaction mining mobile apps”. In: *Proceedings of the 29th annual symposium on user interface software and technology*. 2016, pp. 767–776.

[6] Android Developers. *AdapterView*. 2023. URL: https://developer.android.com/reference/android/widget/AdapterView (visited on 01/01/2024).

[7] Mauro Di Pietro. *Modern Recommendation Systems with Neural Networks*. 2022. URL: https://towardsdatascience.com/modern-recommendation-systems-with-neural-networks-3cc06a6ded2c (visited on 03/08/2023).

[8] Bao Tram Duong. *Fine-Tuning Inputs: Data Preprocessing Techniques for Neural Networks*. 2023. URL: https://baotramduong.medium.com/data-preprocessing-for-neural-network-0b398b43d309 (visited on 12/30/2023).

[9] Alireza Ghods, Diane J Cook. “Activity2vec: Learning adl embeddings from sensor data with a sequence-to-sequence model”. In: *arXiv preprint arXiv:1907.05597* (2019).

[10] Kasthuri Jayarajah, Youngki Lee, Archan Misra, Rajesh Krishna Balan. “Need accurate user behaviour? pay attention to groups!” In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*. 2015, pp. 855–866.

[11] Quoc Le, Tomas Mikolov. “Distributed Representations of Sentences and Documents”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing, Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html.

[12] Seokjun Lee, Rhan Ha, Hojung Cha. “Click Sequence Prediction in Android Mobile Applications”. In: *IEEE Transactions on Human-Machine Systems* 49.3 (2019), pp. 278–289. DOI: 10.1109/THMS.2018.2868806.

[13] Gang Li, Gilles Baechler, Manuel Tragut, Yang Li. “Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale”. In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. https://github.com/google-research/google-research/tree/master/clay. New Orleans, LA, USA: Association for Computing Machinery, May 2022, pp. 1–13. URL: https://doi.org/10.1145/3491102.3502042.

[14] Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, Brad A Myers. "Screen2vec: Semantic embedding of gui screens and gui components". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 2021, pp. 1–15.

[15] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, Jason Baldridge. "Mapping natural language instructions to mobile UI action sequences". In: *arXiv preprint arXiv:2005.03776* (2020).

[16] Google LLC. *Choose a category and tags for your app or game*. 2023. URL: `https://support.google.com/googleplay/android-developer/answer/9859673` (visited on 02/15/2023).

[17] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[18] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, Paolo Cremonesi. "Personalizing session-based recommendations with hierarchical recurrent neural networks". In: *proceedings of the Eleventh ACM Conference on Recommender Systems*. 2017, pp. 130–137.

[19] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, Jianhua Zou. "Deepapp: a deep reinforcement learning framework for mobile application usage prediction". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 2019, pp. 153–165.

[20] Peihuang Wu, Jiakun Zhao. "Distributed representations of html page". In: *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*. IEEE. 2022, pp. 160–164.

[21] Che-Hsuan Yu, Hung-Yuan Chen, Fang-Yie Leu, Yao-Chung Fan. "Understanding mobile user intent using attentive sequence-to-sequence RNNs". In: *Wireless Internet: 12th EAI International Conference, WiCON 2019, TaiChung, Taiwan, November 26–27, 2019, Proceedings 12*. Springer. 2020, pp. 51–61.

[22] Xin Zhou, Yang Li. "Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning". In: *Proceedings of the 15th ACM Conference on Recommender Systems*. RecSys '21. Amsterdam, Netherlands: Association for Computing Machinery, 2021, pp. 473–483. ISBN: 9781450384582. DOI: `10.1145/3460231.3474264`. URL: `https://doi.org/10.1145/3460231.3474264`.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature