Predicting Navigational Patterns in Web Applications using Machine Learning Techniques

Patric Balan

Gustav Jönemo

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6224
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

In large corporations, customer support is a costly service, and an area of constant optimization. Solutions to increase efficiency and decrease bottlenecks are constantly needed. One such bottleneck is support tool proficiency in a customer support organization where many different tools are used, and a potential solution is to let the tool guide the user as it is being used. This thesis explores the use of machine learning to make predictions on user behaviour, based on user web log entries, to simplify the use of a customer support tool for unfamiliar users.

From 1.8 million log entries, four different datasets are created, based on two different data processing principles. On these, three machine learning models are trained, namely an LSTM, a transformer, and a hybrid CNN-LSTM model. These are then compared to a naive baseline model and evaluated on overall accuracy, top-three accuracy, and accuracy based on sequence length.

The results show that all trained models perform better than the baseline model, but not significantly for certain datasets. The trained models also perform very similarly on all datasets, but for long sequences, LSTM generally outperforms the others, reaching an overall accuracy of 75 percent for its best dataset. The respective accuracy for the baseline model is 72 percent. The close results between models can mainly be attributed to the low complexity of the tool from which the log entries originated and the few features they contain.

Acknowledgements

We would like to thank the people in the development team at Ingka Group Digital for assisting us in this project. A special thanks goes to Albin Olsson, our supervisor, for his availability and the help he has provided throughout the project. Additionally, we would like to thank Magnus Petersson, the responsible manager at Ingka, for giving us the opportunity to write this thesis.

We would also like to thank our supervisor at LTH, Johan Eker, for his feedback on the project and the report. Finally, we want to thank our examiner, Karl-Erik Årzén for taking the time to asses this thesis project.

Contents

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
GCP	Google Cloud Platform
LSTM	Long Short Time Memory
ML	Machine Learning
NLP	Natural Language Processing
RNN	Recurrent Neural Network
VM	Virtual Machine
WUM	Web Usage Mining

1

Introduction

1.1 Background

Customer support is a big part of almost any large corporation, and one of the main ways customers interact with the corporation directly. It is one of the greatest factors in customer satisfaction, and therefore has a large impact on the success of the corporation. At the same time, customer support is expensive, and does not directly contribute to the main activities of the company, thus many companies aspire to make their customer support as efficient as possible. This can be done through things such as automated answering machines, customer-selected support categories, self-serving online portals, or chat-bots. What is common between these approaches is that they all happen outside human support interaction, but in most cases, the customer will eventually arrive at a point where they are being directly served by support staff. This is the most expensive stage of the customer-support process for a corporation, and can be the most time-consuming. Making efficiency improvements here is of great benefit both in terms of saved capital for the corporation, but also saved time for the customer, leading to greater customer satisfaction.

At this stage in the customer support process, the support staff usually use some sort of web interface, where they are able to perform a number of actions to help the customer. In a large corporation, and especially one with great ambitions for customer support, the number of actions available to the staff can be of great magnitude. For this reason, it can be a convoluted process to navigate the web interface, especially for newer staff members. Wasted time navigating an interface instead of performing critical tasks is both costly and frustrating for all parties involved. If navigation efforts can be minimized, it can lead to a faster rate of critical tasks performed and thus shorter support ticket processing time.

1.2 The Solution

This thesis proposes a solution for faster and more efficient customer support by predicting users' future actions. This is achieved through the use of Machine Learning (ML) models that analyze data collected from users of a customer support tool. A generic User Interface (UI) is presented in Figure 1.1. Here we can see how a user would interact with the interface, using three different actions enumerated from one to three. By employing machine learning to analyze past user interactions with the interface, one can make predictions about future interactions, such as predicting action three based on one and two.

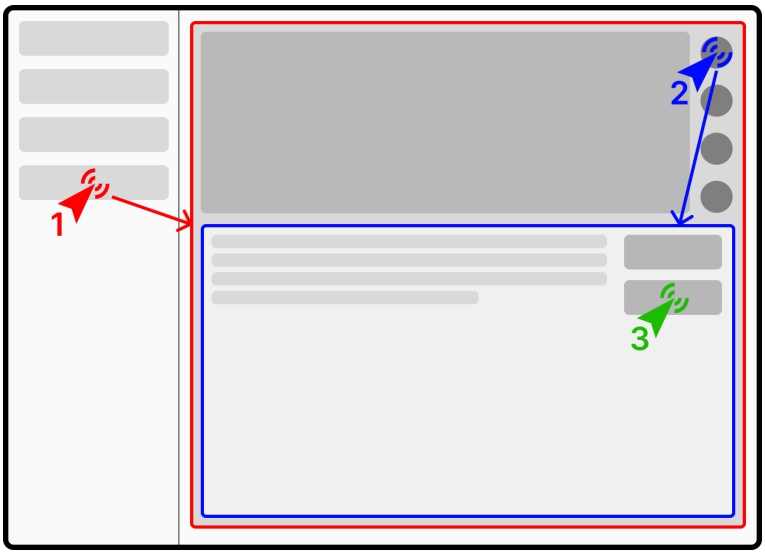

Figure 1.1 There is a certain structure to UI navigation: action one opens a window for action two, which in turn opens a window for action three.

Furthermore, UIs are generally built as tree structures, where only certain information is displayed depending on where in the UI the user currently is. For example, if one were to browse the standard IKEA website in search of a nightstand, it would probably be a subcategory of bedroom furniture, which in turn would be a subcategory of all furniture, placing the user at a tree level of three or deeper. This is called web topology, and by analyzing this structure, more accurate predictions can be made.

As the user interacts with the UI log entries are created, which are files that describe each action taken by the user. These files are then collected and processed. During the processing step, multiple datasets are created, based on if topology is taken

into account, and how far into the future predictions are made. Different machine learning models are then trained on these datasets, after which they are able to make predictions on future user actions.

1.3 The Platform

The platform, henceforth referred to as the customer support tool, is a web tool designed for IKEA's customer support employees as its end users. This platform provides the necessary access and tools for customer support to assist IKEA customers. The tool is also split into two different profiles, one to help private individuals and another designed for companies. The focus of this thesis project will be on the private side.

The home page of the tool presents the user with a search bar for looking up customers. Once a search is executed a new session is created. When a customer is found the user is taken to an overview page. On this page, a list of the customer's recent transactions can be found as well as a menu with options for viewing additional information regarding the specific customer. By navigating through different pages and menus the user has the ability to read and modify information connected to the customer.

The actions performed by the customer support person are continuously logged and saved in the web page front end and later sent in batches at intervals of 30 seconds to the Google Cloud Platform, also known as the GCP[1].

1.4 The Data

The data used for training the ML models consists of several log entries, describing user interaction with the customer support tool. There are five types of log entries: `navigation`, `click`, `change`, `new-user-session`, and `search`. Each log entry type describes a different type of event:

- `navigation` - the user has navigated to a different page, changing the URL
- `click` - the user has clicked on a button
- `change` - the user has used some type of input element, such as a text field or a search bar
- `new-user-session` - the user has opened a session with a customer
- `search` - the user has searched for a new customer

[1] cloud.google.com

Each event contains several attributes with information about the log entries' associated origin. A subset of these attributes is used for training the models, mainly `value, time, sessionId, name`, and `path`. An example of a web log entry can be found in Figure 1.2. A total of 6 million log entries are available, out of which a subset of 1.8 million are used for ML model training.


```
{
  "insertId": "6502ce9f00088f9c9c505d35",
  "jsonPayload": {
    "domain": <DOMAIN_NAME>,
    "value": "modal-list-address",
    "level": 30,
    "consumer": <CONSUMER_ID>,
    "time": 1694682783560,
    "dateTime": "2023-09-14T09:12:17.018Z",
    "sessionId": "cb47ea96-4997-41f7-b862-55c31a0b1bc9",
    "retailUnit": <COUNTRY>,
    "pid": 1,
    "name": "click",
    "coWorkerId": <USER_ID>,
    "timeNumeric": 1694682737018,
    "path": "/customers/{uuid}/profile/information",
    "reqId": "9a04aacd-775f-4b04-9e54-5a6caf477458",
    "hostname": "localhost"
  },
  "resource": {
    "type": "cloud_run_revision",
    "labels": {
      "location": "europe-west1",
      "configuration_name": "co-worker-api-v1",
      "project_id": <PROJECT_ID>,
      "revision_name": "co-worker-api-v1-00015-zul",
      "service_name": "co-worker-api-v1"
    }
  },
  "timestamp": "2023-09-14T09:13:03.561052Z",
  "labels": {
    "instanceId": <INSTANCE_ID>
  },
  "logName": <LOG_NAME>,
  "receiveTimestamp": "2023-09-14T09:13:03.761876598Z"
}
```


Figure 1.2 Example of a "click" log entry from the customer support tool (some values are censored). Each entry is represented as a JSON file containing information about a user interaction.

The log entry attributes contain information about user behavior:

- `value` - the element in the UI associated with the web log entry, if such exist
- `time` - the date and time the log entry was recorded
- `sessionId` - the id of the session initiated by searching for a new customer
- `name` - the log entry type
- `path` - path in the UI to which the action associated with the log entry navigates to, or was performed in

A combination of these log entry attributes is used to filter and group log entries into individual customer support issues, which correspond to sequences that the models will be trained on. Specifically, `value` and `path` are used as input variables, while the rest are used to group sequences.

1.5 Goals

The goal of this thesis is to construct a system that can predict future actions of customer support staff using a web interface to serve customers.

Given an efficient system, this would reduce the number of actions required to perform a given task, thus increasing support efficiency. However, this thesis will focus on the performance of the predictions, and will not dive into the efficiency gains of a customer support operation. System efficiency will be deemed adequate when next-action predictions can be made with 70% accuracy or higher. This number is based on discussions with supervisors with knowledge in the area.

1.6 Outline

The thesis begins with an introduction of the background, solution, and overall goals for the thesis project. Chapter 2 describes the process of Web Usage Mining (WUM), which is the overall approach in which ML is utilized. In the next Chapter 3, an explanation of ML is provided, including more in-depth descriptions of the specific ML models used in the project. This chapter is followed by Chapter 4, which describes the different methodological steps taken during the main activity of the project, and their results. A short discussion is then presented in Chapter 5. The thesis ends with a conclusion and discussion about future work in Chapter 6.

1.7 Contributions

Throughout the project, both authors put in an equal amount of work throughout most parts of the process. Working in parallel, they were always involved in each other's work, even when they were working on different parts of the project. The individual authors did however have their respective focus areas. Patric focused more on the transformer model, in literature and construction. He wrote most of the discussion, and he was also responsible for the WUM literature and construction of the CNN-LSTM hybrid model. Gustav focused more on preprocessing, creating the multiple datasets through data science methods, and he wrote most of the results section, creating the associated graphs while doing so. He also focused more on the LSTM model and ML literature.

2

Web Usage Mining

The process of predicting user behavior based on web logs is a substantial research topic. WUM is the process of extracting useful information from web server logs based on the browsing and access patterns of the users [21]. It is used in many areas such as education, health, and social media, and consists of individual steps that include data preprocessing, pattern discovery, and pattern analysis, all of which will play an important role in this thesis project.

A web log refers to a file where the web server records information whenever a user sends a request to that specific server. The data from a web log can be collected on the client side, server side, proxy servers, or from an organization's database. A log entry example can be found in Figure 1.2.

The WUM process consists of four steps, data collection, data preprocessing, pattern discovery, and pattern analysis [4]. This thesis employs these steps in a modified way, which can be seen in Figure 2.1, with the goal of creating an ML model that can improve user navigation.

2.1 Data Collection

Web data can be divided into four different categories, content data, structure data, usage data, and user profile. Content data consists of information presented to the user, such as text, images, or structured data retrieved from databases. The structure data represents how the web page content is organized. This data may encompass data components utilized within a web page, like HTML or XML tags, or data components utilized in the construction of a website, such as hyperlinks used for connecting one page to another. Usage data includes information such as a user's IP address, time of access, the path accessed, and other attributes that might be included in a web entry log. This is the relevant data used in WUM. User profile data includes all information about the users of the Web site. This information can be collected directly through registration forms and questionnaires or through analyz-

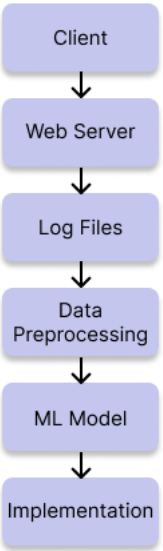

Figure 2.1 A complete WUM process in the context of this project, implementation is however out of scope for this thesis.

ing the web logs [21]. The collected data can come from the client side, server side, proxy servers, or an organization's database.

2.2 Preprocessing

Because the collected data from the web server is often unstructured or incomplete, data preprocessing is a necessary step in the WUM processes. The data preprocessing task may include steps such as data cleaning, user identification, user session identification, path completion, and data integration [23].

Data Cleaning

During data cleaning items that are not relevant for usage analysis are removed from the log files. If the web log entry contains information such as images, videos, scripts, and other irrelevant information in the context of usage data it should be removed. Additionally, if the status code of a log entry is outside the span of 200 to 299, signaling an error, it should also be removed [21].

User Identification

Once the data is cleaned the next step is to identify individual users of the platform. This can be achieved through login information, assuming the user logged in before using the website, or with the information gathered from cookies [21]. Another approach is to study the IP address and assign a different ID to a different IP address. However, different users may share the same IP address and one user may use many different browsers. To combat this both IP address and user agent are checked to identify a new user [23].

Session Identification

After a user is recognized, the series of clicks made by each user is grouped into coherent clusters, also called sessions. The two main approaches for identifying user sessions depend on time and navigation in web topology.

The most basic approaches are time-centric, with one method relying on the overall session duration and the other on single-page stay time. The collection of pages visited by a user in a specific time interval is called viewing time and it may vary from around 25 minutes to 24 hours. The other time approach looks at how much time has passed between two log entries, if the timeout threshold has been exceeded the next entry is considered to be a new session. Time-based methods are considered to be efficient but unreliable since the user may be involved in other activities while browsing the web page.

The use of web topology is another approach when dealing with session identification. This approach looks at web page connectivity without the need for a hyperlink between the two following pages. When a web page is not linked to a previously visited page within a session, it is categorized as a separate session [23][4].

Path Completion

Because of page caching and proxy servers, some key accesses are not recorded and are missing from the web server log. Because of this, the access patterns of users are not represented accurately in the access log. To combat this, HTTP referrer information in server logs, and knowledge of the website structure can be used for path completion. Some conclusions can be reached by looking at if a user has requested a page that is not directly linked to the last page. If that is the case the referrer log can be checked to see from which page the request came. Furthermore, if the page is in the user's recent request history, it is presumed that the user has backtracked using the backtracking feature in the browser. The website structure can also be used if the referrer information is ambiguous. In the case that two or more different web pages include a link to the requested page, the closest page is considered the source of the most recent request. This approach allows for the identification and inclusion of missing page references in the user sessions [4].

2.3 Pattern Discovery

Once the data has been preprocessed it is ready for pattern discovery. There are many different techniques used for pattern discovery such as discovery of association rules, sequential patterns, clustering, and classification[21].

Classification is a supervised ML process that uses predefined categories. Should there be a need to create individual profiles describing a particular user belonging to a category or class then features can be extracted to describe those particular properties. Some of the most popular classification algorithms are decision trees, neural networks, Bayesian classifiers, and probability theory for classification [23].

Finding association rules means looking for what set of pages are frequently accessed together, which page will be accessed next, and the overall correlation between actions on the website. If two pages are frequently viewed in correlation then it might be efficient to interconnect those two pages.

Clustering is used to group users based on shared characteristics. Users who share similar browsing patterns can be clustered together. This is particularly useful in e-commerce when trying to identify different types of users to provide personalized services and recommendations.

Sequential patterns are studied to find patterns within a session where the next action can be predicted based on the sequence of previous actions. With this information smart recommendations based on a predictive model can be presented to the user [23][21].

2.4 Pattern Analysis

The final step in WUM is pattern analysis where uninteresting patterns or rules are filtered out. Some techniques for pattern analysis are data and knowledge querying, online analytical processing techniques, and usability analysis. The outcome of pattern analysis improves the system performance and changes the website. It contributes to bringing in visitors and providing a personalized experience to established users [23].

Personalization of the user experience is one of the applications of WUM. With the increasing complexity and possibilities of web browsing increasing each day, the need for a personalized user experience increases as well. Web penalization aims to align with the user's behavior and interests when navigating the web. An example of personalization includes recommender systems which aid the user in navigating the web page by recommending additional products or future actions.

E-commerce is another application of pattern analysis since two of the main goals of

web-based companies are customer attraction and customer retention. The insights gathered from WUM can aid the web-based company to improve both of those goals and even improve the design of the web page.

Another application of WUM is prefetching and caching. Prefetching entails the loading of the data before it is required to decrease the wait time. With the use of the patterns discovered from the previous step, prefetching and caching strategies can be employed to reduce the server response time [7].

3

Machine Learning

"Machine learning is a branch of artificial intelligence and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy" [15].

Furthermore, ML is about letting machines discover their own algorithms using data points that are characterized by various properties. These data points can be subdivided into features and labels, depending on whether they can be easily computed or if they require human identification, respectively [1]. The primary goal of ML is to learn to predict the label of a data point based only on its features [17].

3.1 Learning Methods

The use of these data points varies depending on what type of ML method is used, where the main methods are supervised, unsupervised, and reinforcement learning. They differ in several ways, mainly in the type of data provided for the training of the algorithm. In supervised learning, the training data consists of input-output pairs, where the output is the desired label for the input. In contrast, unsupervised and reinforcement learning does not have access to such labeled data during the training phase. As a result of this, unsupervised methods need to rely only on the inherent structure of data points to learn. In the case of reinforcement learning, reward systems are also used to increase the precision of the predictions by providing the ML model with input on prediction accuracy as it learns [17].

Self-supervised learning is another learning technique that uses unlabeled data to generate labels for the purpose of training a model. The employed strategy consists of using the relationships or structure within the data to create useful learning objectives. This learning algorithm can for example learn to predict the missing parts of a sentence by using the available data as both input and output. This type of learning closely resembles how humans learn from the world, without requiring explicit labels or instructions [3].

3.2 Metrics

In order to train and evaluate ML models one needs to define clear metrics to determine what constitutes a good result. A common way for ML methods to determine the level of improvement between training steps is by using loss functions. They produce a score describing how far away a prediction was from the true value, where 0 would be a perfect score. One such function is the cross-entropy loss function, also known as logarithmic loss, which produces logarithmically higher penalties the further away from the true value the predictions are. As the model is being trained, it tries to minimize the loss function, and as long as the loss decreases, the model is improving and therefore learning [18].

While loss is an appropriate metric for training ML models, it does not necessarily portray an intuitive picture of the performance of the model. For this purpose, accuracy can be more relevant. Accuracy is calculated by taking the number of correct predictions divided by the total number of predictions and therefore produces a score describing how often the model makes correct predictions.

Accuracy is binary, in the sense that a prediction is either right or wrong. In this project, in addition to traditional accuracy, accuracy is also calculated on multiple close predictions. Meaning if any of a number of potential predictions are correct, it will count as a successful prediction. This is important in the context of user interaction, where the user might benefit from several predicted options to choose from.

3.3 Artificial Neural Networks

An Artificial Neural Network (ANN) is a type of ML method with interconnected nodes aiming to simulate biological neural networks such as the ones in animal brains. In its most basic form, it consists of three layers: an input, hidden, and output layer. The hidden layer often consists of many interconnected layers, making the ANN a Deep Neural Network (DNN), as illustrated in Figure 3.1 [2].

Each node in the network acts as its own linear regression model with an input, weight, bias/threshold, and output. Given an input and weight, if the output reaches a certain threshold determined by an activation function, the node fires and passes on data to the next layer [12].

3.4 Deep Learning

Deep Learning employs the use of DNN to learn more complex and abstract features from data, and in turn, achieve better performance. The DNNs are used for both supervised and unsupervised tasks, depending on the type of available data.

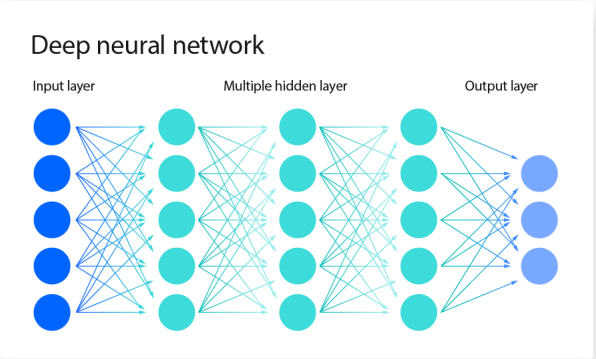

Figure 3.1 A DNN is an ANN with many hidden layers, which allows the network to make more complex decisions [12].

Deep learning algorithms excel at processing unstructured data like text and images while also automating the feature extraction process. The process in which the DNN uses its interconnected layers to gradually improve the prediction or categorization is called forward propagation. Back-propagation is the process in which algorithms like gradient descent are employed in order to calculate and adjust errors in the prediction by moving backward through the layers. By learning from the data and correcting its own errors, a neural network can enhance its performance and accuracy [14].

3.5 Gradient Descent

The gradient descent optimization algorithm aims to minimize the cost function of a model. The cost function's gradient represents the derivative concerning the model's parameters and indicates the direction of the steepest ascent. The algorithm updates the parameters in small steps to minimize the cost function. For each iteration, a gradient of the cost function is computed which indicates the direction of the steepest ascent. By moving in the opposite direction of the gradient the steepest descent is found. The speed of how quickly the algorithm moves toward a minimum is determined by the learning rate. This process can be repeated until the cost function converges to a minimum, which indicates that the model has reached an optimal set of parameters.

There are three main types of gradient descent. Batch gradient descent updates the model's parameters by using the gradient of the entire training set. This aims to converge towards a global minimum, however, it may be computationally expen-

sive and slow when dealing with large datasets. The second version of gradient descent, stochastic gradient descent, uses the gradient of one training example at a time. Stochastic gradient descent is computationally efficient and can converge faster than batch gradient descent but can prove noisy and may not converge to the global minimum. The third type of gradient descent combines the advantages of both stochastic and batch gradient descent. Mini-batch gradient descent updates the model's parameters using the gradient of a small subset of the training set. With this strategy, it is computationally efficient and less noisy than stochastic gradient descent, while still being able to converge to a good solution [6].

3.6 LSTM

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) model used to predict sequential data. Its main advantage over traditional RNNs is that it can process longer sequences while maintaining accuracy [10].

RNNs are a type of ANN or, more commonly, DNN, differing only in that the nodes of an RNN also map onto themselves, giving the network memory capabilities. This makes RNNs suitable for sequential data, where a future data point often will rely on a previous one. However, a drawback in RNNs lies in how this memory is passed on between layers. The data is rounded as each step, meaning that for long sequences, the memory tends to diminish or explode into infinity.

LSTM on the other hand does not have this disadvantage. LSTM is a type of RNN with the differentiating property that each node consists of a cell with three gates; an input, output, and "forget" gate, controlling the data flow in the network. This setup allows the cells to selectively pass on memorized data, resulting in better state awareness and thus enabling the prediction of longer sequences [10][13].

3.7 Convolutional Neural Network

A convolutional neural network (CNN) is a type of network architecture for deep learning that can process multidimensional data, such as images, and audio signals. This is done through the convolution linear operation that involves the multiplication of a small region of the input data, called kernel or filter, with a set of weights and then summing up the result to produce a single output value. By performing this operation throughout the input data a feature map is obtained. The feature map acts as the new representation of the input data, highlighting features that the kernel detected.

A CNN is comprised of three main types of layers. These layers are the convolutional layer, the pooling layer, and the fully-connected layer. As the name suggests, the convolutional layers apply convolutions to the input data, using multiple kernels

to extract features. The pooling layers are used to reduce the size and complexity of the feature maps. This is done by applying a downsampling operation such as taking the maximum average of a region. Fully connected layers act similarly to the layers in a standard feed-forward layer, where each neuron is connected to all the neurons in the previous layer. The fully connected layers are used to perform the final classification or regression task based on the features extracted by the previous layers.

CNN uses a supervised learning algorithm to learn the optimal weights for the kernels and the fully-connected layers. The learning algorithm adapts the weights based on the error between the predicted output and the label, using a loss function. The error is propagated back through the network using back-propagation to minimize the loss function.

Computer vision tasks such as image classification, object detection, face recognition, and scene understanding are best suited to CNN. However, the CNN can be applied in other domains, such as NLP, speech recognition, and audio synthesis [11][8].

3.8 Natural Language Processing

NLP refers to the branch of AI that allows computers to understand natural language as produced through speech or text by humans. The algorithmic approach to archiving this has evolved over time, with early approaches focused on processing data by applying specific rules. Today, NLP is mainly conducted through the use of ML, which uses large datasets in order to create statistically powered models. [16].

Word embedding is often used in NLP problems in conjunction with ANNs. It is the technique of mapping words to vectors of numbers, as illustrated in Figure 3.2. The idea is to capture the meaning of words in these vectors so that two different words with similar meanings will result in similar vectors. This allows the model to understand the text it is given to some extent, while also feeding the ANN a numerical representation of the text which is easier for the network to process [25].

Tokenization is the process in which a sequence of words is broken up into smaller units called tokens. By doing this a neural network can process text as individual words, characters, or even sub-words. For example, the sentence "Hello world" can be split into two individual word tokens "Hello" and "world" [22].

The vocabulary in NLP refers to the set of unique tokens in the corpus. Sometimes only a set of the most frequent tokens in the corpus is considered when building a vocabulary. Traditional NLP approaches use the vocabulary tokens as unique features. In contrast, more advanced deep learning NLP approaches use the vocabulary

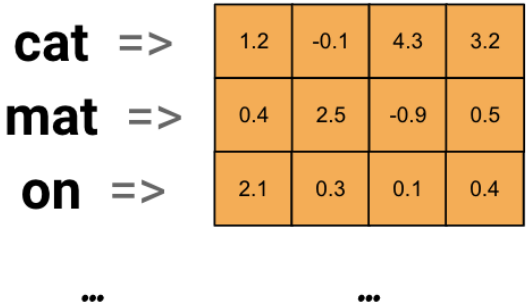

Figure 3.2 Example of a 4-dimensional word embedding [9]. Embeddings enable complex representations of words which improves learning in ML models.

to create the tokenized input sentences and later use the tokens of the sentence as inputs to the model [22].

3.9 Transformers

A transformer is a deep neural network (DNN) model that employs the multi-head attention mechanism. This model was first proposed by Google in 2017 where it showed promising results in NLP tasks such as translation and constituency parsing. Instead of relying on recurrence like the LSTM model, the transformer model avoids it entirely and instead employs the use of the attention mechanism to establish dependencies between the input and output [26].

Attention was created to improve the performance of the encoder-decoder model in machine translation, by using a weighted combination of all the encoded input vectors, granting the most relevant vectors the highest weights [5]. This is contrary to RNN which favors neural activation for more recent tokens at the end of a sequence. Attention allows all tokens equal access at any part of a sequence while doing so in parallel to increase the processing speed [20].

Self-attention is the attention mechanism that considers multiple positions within a sequence to generate a representation of the entire sequence. Regarding computational complexity, self-attention shows better performance than recurrent layers when the sequence length is smaller than the representation dimensional, which is often the case with sequence representations used in state-of-the-art models [26].

The multi-head attention consists of several attention layers running in parallel [26]. This allows the model to capture different types of dependencies between the input and output, which is useful for tasks such as machine translation and language modeling. The multi-head attention mechanism is more powerful than the single-head

attention mechanism because it allows the model to attend to different parts of the input sequence with different learned parameters, which can capture more complex relationships between the input and output. [19].

4

Method & Results

The methodology involves the use of existing user data for model training and development. The collected data comes from IKEA's customer support in the form of unlabeled log entries of the actions performed by the support staff when helping customers. These actions are performed on an in-house web tool designed for customer support and are continuously logged and saved in JSON format.

The first step in the process is focused on research of relevant literature involving potential methods for training prediction models in the context of WUM.

After conducting the research, the following task involves examining the raw data. Since web applications are built as trees, the traversability of the web tool's tree structure is analyzed and visualized graphically to gain an understanding of the dataset.

The collected data then undergoes a series of processing steps before it can be used. First, log entries are sorted chronologically and by session, yielding sequences of actions for each customer support issue. Then, redundant and invalid log entries are filtered out, and subsequently, sequences that are of insufficient length.

With a suitable dataset established, model training can begin. After the training phase, the models are validated, and based on the validation some fine-tuning is conducted to improve the models. Finally, the improved models undergo a final evaluation and the results are presented.

4.1 Environment Setup

Data processing and model prototyping were mainly performed on local machines, using Python and a remote git repository. This approach was feasible for small datasets, but as the amount of data grew the power of the local machines was insufficient. To train multiple models with different hyperparameters on large datasets, Virtual Machines (VM) in the cloud were utilized. These were provided by IKEA

which uses the GCP, which offers a number of cloud computing options. For this project, the GCP Vertex AI service was deemed most appropriate, as it provided a VM and service tailored for ML tasks, with options such as additional graphics cards and built-in JupyterLab support. Through the Vertex AI service, it was possible to directly interact with the VM and execute code remotely without any need for containerization or external storage.

As for libraries, Keras[1] with a TensorFlow[2] backend was used for ML tasks. Additional utilities were provided through NumPy[3] and scikit-learn[4], and illustrations were created using Matplotlib[5].

4.2 Data Collection

To perform the experiments, web usage log entries from the customer support employees spanning several months were collected from the IKEA cloud server. These entries were accessed using queries on Splunk[6], a cloud-based log hosting service. The acquired log entries are structured in JSON format.

To get a complete picture of user behavior only the "click" and "navigation" entries were selected since the "new-user-session" and "search" entries did not provide additional relevant information in reconstructing the usage of the customer support web tool. "Change" entries were initially also used in an attempt to bring more information about a session but proved to introduce more uncertainty to the predictions made by the ML model. Additionally, only entries containing a session ID were filtered on since about a third of all recorded entries contained no information except for the "name" field. This was a result of accessing the customer support tool from an external program. Thankfully, the incomplete entries belonged to separate sessions in the log and could be filtered out without the need for path completion. A little over 6 million entries were collected from the database, out of which 1,8 million were used to train the models.

Data Processing

Even though the collected data was structured in the JSON format it had to undergo some processing first. Initially, the data had to be stripped out of the raw JSON log entries down to a stream of individual log entries with the corresponding attributes. Once all the data contained the relevant information it could be divided into separate

[1] keras.io
[2] tensorflow.org
[3] numpy.org
[4] scikit-learn.org
[5] matplotlib.org
[6] splunk.com

sessions using the "sessionId" attribute. Each session was ordered into the correct sequences using the "timeNummeric" attribute.

Once the data was separated into individual sessions and ordered chronologically it required additional trimming. Each session started with the same log entry sequence which does not provide relevant information for training a model. Similarly, the end of most sequences ended with the navigation to the home page which was also removed. Additionally, it was discovered that some log entries were duplicated and also had to be removed. Lastly, specific log entries were removed because they did not provide additional information about the navigation and usage of the web tool. After this additional trimming of the sessions, all sessions containing only one log entry were removed.

For the "click" log entries, information from the "path" and the "value" attributes was combined to identify the unique action performed by the user. This is due to some elements located on different web pages sharing the same "value". For the "navigation" log entries only the "value" attribute was required since both the path and value attributes are identical for this type of log. Special characters like "/" and "-" were removed. An example of a processed log entry in the corpus after the filtering of special characters and the combination of its value and path attributes looks like this: " customersuuidactivityrewardssendvoucher". Here we can see the path "cutomers/uuid/activity/rewards" at the start and the value "send-voucher" added to the end. A sequence in the corpus consists of multiple log entries processed to look like the example provided. This way all different UI actions have a unique identifier. By the end of the data processing, 116 unique log entries could be identified.

After the preprocessing was completed a total of 90530 sequences could be used, where the average sequence length was 4.7 actions.

Web Topology

All websites have a topology, a tree structure that describes the pathways of the website. The customer support tool is no different, and from this topology, information can be extracted that helps increase accuracy in predictions. The main reasoning is that actions performed in different parts of the tree are unrelated, and not treating them as such will introduce uncertainty in the predictions. For example, signing up a customer for weekly ad emails is probably not related to issuing a refund. Therefore it makes more sense to treat these two issues separately.

To take this into account, an alternative dataset was created where sequences containing subsequences in different parts of the tree were split into smaller sequences, based on the web topology. This can be seen in Figure 4.1, where we can observe a longer sequence of actions being split into smaller sequences. Actions that result in a branch change or a path that traverses in the opposite direction of the tree structure are discarded, and thus only sub-sequences 1 and 3 are considered.

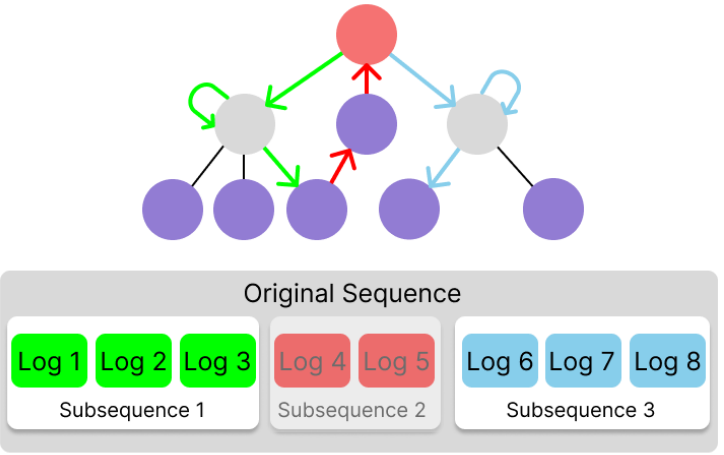

Figure 4.1 Long sequences of actions can be divided into shorter ones based on topology, which allows for greater prediction accuracy.

N-grams

N-grams are used to create smaller sequences out of larger ones, as illustrated in Table 4.1. Using this method, models are able to make predictions on what kind of log entry (and thus what kind of user action) will follow a given sequence, regardless of sequence length.

Sentence		Log entry sequence	
Predictors	Labels	Predictors	Labels
The	quick	click1	click2
The quick	brown	click1 click2	nav1
The quick brown	fox	click1 click2 nav1	click3

Table 4.1 N-grams example showcasing similarities between sentences and sequences of log entries. Comparison between the sentence "The quick brown fox" and the sequence "click1 click2 nav1 click3".

Two types of n-grams are used, next-action and last-action. Next-action n-grams are created as illustrated by Table 4.1, where the resulting sub-sequences represent the next action in a sequence of actions. Last-action n-grams differ in that all sub-sequences end with the same value, which is the last value of the original sequence. These sub-sequences aim to train the models to predict the last action in a sequence of actions instead. Both types of n-grams are illustrated in Figure 4.2.

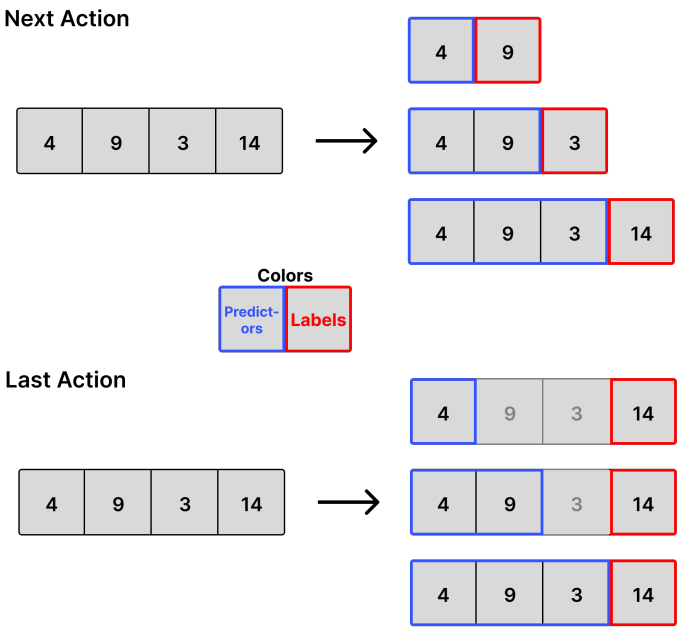

Figure 4.2 Next-action n-grams are suited for predicting the next action in a sequence, while last-action n-grams are suited for predictions far into the future.

Using the two types of n-grams and topology splitting, smaller sequences were created from the processed data. The final four datasets as a result of the data processing can be seen in Figure 4.3.

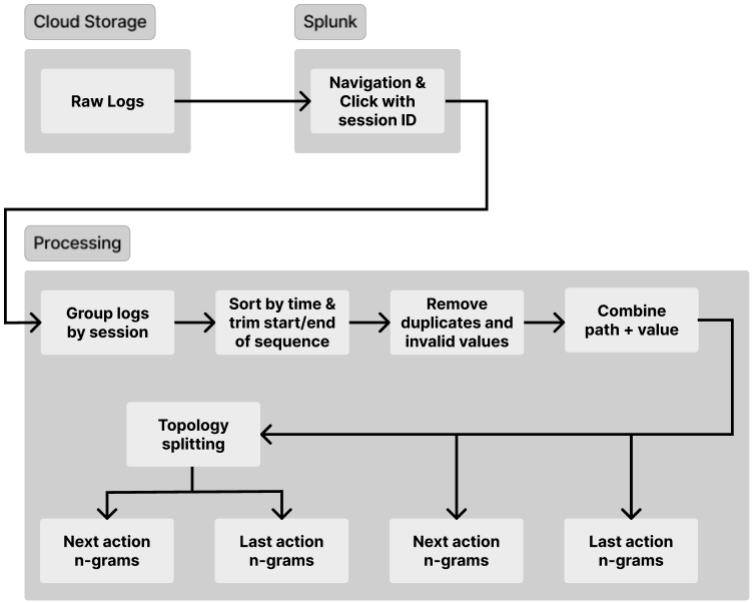

Figure 4.3 After a series of data collection and processing steps, four different datasets emerge.

4.3 Tuning & Validation

Many adjustable hyperparameters can be set for each model, all of which affect the model behavior. Setting the value of these hyperparameters is not trivial, and the best values for a given task can be difficult to achieve. A common approach to finding good values is by employing automatic hyperparameter tuning, where combinations of certain hyperparameter values in different ranges are constructed and then used to train the model.

From experimentation, three main hyperparameters were identified as having a particularly significant effect on model performance. These were *dropout*, *learning rate* and *batch size*. The dropout parameter controls the dropout rate of neurons during training, with the goal of reducing the overfitting of the model on the training data. Learning rate controls the rate at which the model adjusts its weights. Finally, the batch size parameter defines the number of samples to learn from before updating the internal model parameters. Knowing this, 36 combinations were created from these hyperparameters, with the values outlined in Table 4.2.

Out of the four datasets that resulted from the data processing, two were used in combination with these parameters, namely the two next-action sets. When these

Hyperparameter	Values
`dropout`	0.1 0.3 0.5
`learning rate`	0.01 0.001 0.0001
`batch size`	1 16 32 64

Table 4.2 Hyperparameter values tested to maximize model accuracy, first set (36 combinations).

tuning jobs had finished, we realized certain values consistently underperformed and adjusted them accordingly. The new values were used for the remaining two last-action sets and can be seen in Table 4.3.

Hyperparameter	Values
`dropout`	0 0.1 0.3
`learning rate`	0.01 0.005 0.001
`batch size`	16 32 64

Table 4.3 Hyperparameter values tested to maximize model accuracy, revised set (27 combinations).

For each hyperparameter combination, the models are evaluated by passing the test data to the trained model. The model outputs its prediction for the next action in a sequence, or the last, depending on training mode, for each sequence. The prediction is compared to the label and an accuracy score is calculated by dividing the number of correct predictions by the total number of predictions. This metric is used because it works well in representing the performance of multiclass classification problems.

The best-performing model versions are further evaluated by their top-3 score and depth performance. The top-3 score is calculated similarly to accuracy, but at each step, the model produces the three most likely predictions instead of one, and if any match the label the prediction is considered a success. This evaluation method mimics a scenario where the customer support agents are presented with multiple predicted options in the UI.

Depth performance is a measure of how well the models perform in relation to the distance between predictors and labels in the case of last-action prediction, and sub-sequence depth, i.e. the number of predictors used for each prediction, in the case of next-action prediction. An illustration can be found in Figure 4.2. Here, the predictor count is 1, 2, and 3, counting from the top for both next- and last-action n-grams. For last-action n-grams however, the label is always the last symbol of the original sequence, and thus a distance between predictors and label occurs, in this case 2, 1, and 0, counting from the top. Each model is evaluated on depth performance, and the results can be seen in Section 4.8.

4.4 Baseline

Architecture

To learn more about the data and have a reference point for future improvements, a custom-built baseline algorithm was used for predicting the next token in a web browsing sequence. The baseline algorithm started as a weighted random classifier, able to choose between all the possible following actions for each action in a sequence. Another version of the baseline, which outperforms the weighted random approach, takes the same probability distribution but only selects the next token with the highest probability. The next action is selected using the information gathered from the training set. For each token in a sequence, the next action is selected from a probability vector belonging to the current token containing all the possible following tokens. This is instead of ANNs which are used in the other, deep learning models. A limitation of this approach is that the baseline can only make next-action predictions based on only one predictor, and is thus absent in the performance graphs in Section 4.8. The algorithm is tested on a test set containing new sequences. Out of the two versions of the baseline algorithm, the best-performing one was chosen to evaluate the improvements made by the ML models.

A visual representation of how the baseline model functions can be seen in Figure 4.4. Each unique token in a sequence has its own probability vector which is used in the selection of the next token.

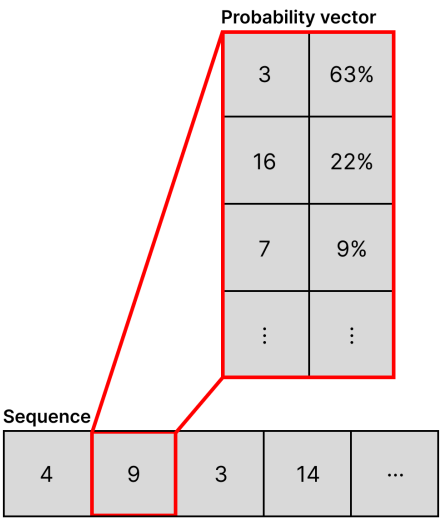

Figure 4.4 The baseline makes predictions on the next action based on the unique action ID. The most likely ID to follow is chosen.

Results

The results from the baseline next token prediction on both the split and non-split dataset can be seen in Table 4.4. Last-action data is unavailable since the probability vector of the baseline only keeps track of next-action tokens.

Baseline	Acc (%)	Top-3-acc (%)
Next	49.60	78.02
Next Split	71.75	96.96

Table 4.4 Results from the Baseline. The baseline performance is shown using the accuracy metric in making correct predictions on different datasets.

4.5 LSTM

LSTM is a popular ML model for sequential tasks, such as predicting future values from time series data or predicting the next word in a sentence. The latter is often the case in NLP, and since the log values can be described as tokens much in the same way words can, it was deemed appropriate to approach the user interaction prediction problem as an NLP problem using LSTM. But, instead of using words in a sentence, log entries in a log sequence are used.

Architecture

The LSTM architecture employs the use of the Keras Tokenizer class to convert the input text into numerical sequences of tokens. Each token represents an action in the web usage sequence. With the help of the tokenizer, a vocabulary is created containing all the unique tokens, and each token is assigned an index. The LSTM architecture takes in n-gram sequences that later get padded with empty values so that all sequences become the same length, which is necessary for the input layer in the LSTM architecture.

The embedding layer, which also acts as the input layer for the model, automatically creates an embedding from the input sequences using the Keras embedding layer class. The embedding layer connects to a dropout layer that randomly sets input units to 0 in order to prevent overfitting. Following the dropout layer comes the LSTM layer containing several hidden layers. After the LSTM layer, there is an additional dropout layer. The output layer is a dense layer that maps the hidden state vector to a vector of the same size as the vocabulary, using a softmax activation function. This layer produces the probability distribution over the possible predicted token. The model is then compiled using a categorical cross-entropy loss function and an Adam optimizer.

Early versions

Early versions of the LSTM architecture evolved in parallel with the preprocessing, with the initial focus on making next-action predictions using unfiltered click log entries. While accuracy was high, this method was not representative of real-world use and would therefore be impractical. To make more useful predictions, higher log granularity was required, and thus navigation log entries were added. At this point, more useful predictions could be made at the cost of lower accuracy. However, the preprocessing for the early versions did not account for different log entries having the same "value" attribute. To combat this, "path" and "value" attributes had to be combined to create unique tokens for each type of log entry in the customer support tool. This decreased accuracy, but once again provided more realistic predictions.

To increase accuracy, experiments using bidirectional LSTM layers, sliding windows, and bi- as well as trigrams were conducted, but the results were unfruitful. The next step was to increase sequence fidelity further by also including "change" log entries. While this resulted in longer sequences which in theory would allow for better predictions, the chaotic nature of these log entries reduced precision. Changes are less sequential than navigation and clicks, in the sense that when multiple changes are made at once they are made without any clear order. Such as when changing the address information of a customer where five different fields can be changed in any order. "Change" log entries were subsequently removed, but the model was expanded to also predict the last action in a sequence, in addition to next-action predictions.

Results

The results of four different LSTM model variants are presented in this section. Two of them use the topology split dataset and the other two do not. The difference between the two model variants for each dataset comes from what type of prediction they are making. One model variant predicts the next action in a sequence while the other predicts the last action in a sequence. Each model is evaluated using the accuracy for both its most likely prediction and its top-3 prediction. The accuracy achieved with the different LSTM model versions can be seen in Table 4.5.

LSTM	Hyperparameters (Dropout - LR - BS)	Acc (%)	Top-3 acc (%)
Next	0.1 - 0.0001 - 64	55.03	82.14
Next Split	0.3 - 0.001 - 64	75.35	96.00
Last	0.3 - 0.001 - 64	38.18	63.89
Last Split	0.1 - 0.005 - 16	62.28	88.81

Table 4.5 Results from the LSTM model variants. The models are compared based on their accuracy in making correct predictions on different datasets.

4.6 CNN-LSTM Hybrid

Both CNN and LSTM are used in NLP problems. The CNN is more powerful when it comes to learning patterns in data while LSTM is more efficient at capturing long-term dependencies in sequential data. In an attempt to improve the performance of both of these neural networks, a hybrid network was used [24].

Architecture

The CNN-LSTM architecture uses the tokenizer class from Keras to convert the input text data into numerical sequences of tokens. Each token represents an action in the web usage sequence. With the help of the tokenizer, a vocabulary is created containing all the unique tokens, where each token is assigned an index. The tokenized sequences are split into n-grams and padded.

The CNN-LSTM hybrid architecture uses a custom layer designed to combine the benefits of both the CNN and the LSTM architectures. This layer consists of three sequential sublayers. The CNN layer is the first sublayer, followed by an LSTM layer, and finally a fully connected layer. The CNN layer consists of two iterations of a convolutional layer followed by a pooling layer. The LSTM sublayer is comprised of an LSTM layer, containing multiple hidden layers, followed by a dropout layer to combat overfitting. The last sublayer is the fully connected layer.

The main architecture consists of an input, embedding, CNN-LSTM, dropout, and output layers. The input of tokenized integers is passed through an embedding layer that converts each token into a vector of a predetermined size. The embedding layer connects to the CNN-LSTM custom layer which uses the output from the CNN as an input to the LSTM, allowing the LSTM layer to learn features from the input data that have been learned by the CNN. The output from the CNN-LSTM layer is passed through a dropout layer to prevent overfitting. The final output layer is a dense layer with a softmax activation.

The model is trained and evaluated using the Keras fit and evaluate methods. The data split into training and test sets is done using the sklearn's train_test_split function.

Results

In this section, four different CNN-LSTM model variants are presented. Two of them use the topology split dataset and the other two do not. The difference between the two models for each dataset comes from what type of prediction they are making. One model predicts the next action in a sequence while the other predicts the last action in a sequence. Each model is evaluated using the accuracy for both its most likely prediction and its top-3 prediction. The accuracy achieved with the different CNN-LSTM model versions can be seen in Table 4.6.

CNN-LSTM	Hyperparameters (Dropout - LR - BS)	Acc (%)	Top-3 acc (%)
Next	0.3 - 0.0001 - 16	54.76	81.87
Next Split	0.1 - 0.001 - 64	75.41	95.97
Last	0 - 0.0001 - 64	38.25	64.15
Last Split	0.1 - 0.001 - 64	62.01	88.70

Table 4.6 Results from the CNN-LSTM model variants. The models are compared based on their accuracy in making correct predictions on different datasets.

4.7 Transformer

The transformer learning architecture was a natural continuation of the LSTM model based on the advantages it has over RNN architectures. Transformers have also been successfully used for NLP and sequence-to-sequence tasks which makes this architecture notable for further experimentation.

Architecture

The transformer architecture uses the Keras Tokenizer class to convert the input text data into numerical sequences of tokens. Each token represents an action in the web usage sequence. With the help of the tokenizer, a vocabulary is created containing all the unique tokens, each token is then assigned an index. The tokenized sequences are split into n-grams and padded.

A custom transformer layer that implements the core principles of the transformer model was created for this architecture. This layer consists of two sub-layers, a multi-head attention layer followed by a feed-forward network. The multi-head attention layer computes the self-attention scores of the tokens in the input sequence and produces a weighted sum of the input embeddings. The feed-forward network adds two dense layers with ReLU activation and a dropout layer. The transformer block also uses layer normalization and residual connections aiming to stabilize the training and improve performance.

The main architecture of the model connects the input, the transformer block, and the output layers. The input sequence of token indices is passed through an embedding layer that converts each token into a vector of a predetermined size. The embedding layer also adds positional encoding to the embedding to capture the order of the tokens. The output from the embedding layer is passed to the transformer block, which in turn produces a sequence of vectors that represent the input tokens with self-attention. The output from the transformer block is fed into a global average pooling layer that reduces the sequence to a one-dimensional vector. This vector is then passed through a dense layer with ReLU activation and a dropout layer. The final output layer is another dense layer with a softmax activation that predicts the

probability of each token in the vocabulary as the next word in the sequence. The model is compiled using a sparse categorical cross-entropy loss, an Adam optimizer, and a sparse categorical accuracy metric.

The model is trained and evaluated using the Keras fit and evaluate methods. The data split into training and test sets is done using the sklearn's train_test_split function.

Results

The results of four different transformer model variants are presented in this section. Two of them use the topology split dataset and the other two do not. The difference between the two models for each dataset comes from what type of prediction they are making. One model predicts the next action in a sequence while the other predicts the last action in a sequence. Each model is evaluated using the accuracy for both its most likely prediction and its top-3 prediction.

The accuracy achieved with the different transformer model versions can be seen in Table 4.7.

Transformer	Hyperparameters (Dropout - LR - BS)	Acc (%)	Top-3 acc (%)
Next	0.1 - 0.001 - 64	54.08	81.79
Next Split	0.1 - 0.001 - 64	75.41	95.99
Last	0.1 - 0.005 - 32	36.70	62.92
Last Split	0 - 0.001 - 16	62.36	89.17

Table 4.7 Results from the transformer model variants. The models are compared based on their accuracy in making correct predictions on different datasets.

4.8 Depth Performance & Model Comparison

In this section, a comparison between models is presented. A separate comparison is made on each of the four datasets on which the models are trained, and the graphs presented represent their performance on these in relation to predictor count and label depth. This shows how performance varies depending on sequence length and how far into the future the model needs to predict.

In Figure 4.5 on page 42 the next-action accuracy for different predictor counts can be seen. It can be observed that as the predictor count increases, the variation in accuracy increases. This is a result of data deficiency since there are significantly fewer long sequences than there are short ones, keeping in mind the average sequence length for this dataset is 4.7. At the end of the graph, a point is reached

where there is only one sequence to evaluate on for a certain length, and thus accuracy is either 1 or 0.

For reference, the overall accuracy of all models on this dataset is around 54-55 percent. This can be compared to the early peaks that exceed 60 percent accuracy, and the early low points that dip below 50 percent accuracy. For these points, all models perform similarly, but for sequences with more than 3 predictors, LSTM outperforms the others.

In Figure 4.6 on page 42 the same next-action accuracy for a different number of predictors can be seen, this time for the dataset that has been processed based on topology. As the topology processing involves division of longer sequences the average sequence length decreases to 2.85, and the point of data deficiency occurs earlier, resulting in overall shorter sequences. For this dataset, the overall accuracy is around 75 percent, which can be compared to the peaks and low points of 80-90 percent and 60-70 percent respectively. LSTM once again outperforms the others for longer predictor counts.

In Figure 4.7 on page 43 the last-action accuracy for different label depths can be seen. Overall accuracy for this dataset is around 36-39 percent between the different models. It can be observed that accuracy quickly drops before reaching an equilibrium at around 30 percent accuracy. At deeper depths, accuracy drops exponentially until it reaches 0. The overall best-performing model is once again LSTM.

In Figure 4.8 on page 43 the same last-action accuracy can be seen, now evaluated on the topology processed dataset. The overall accuracy for this dataset is around 62 percent. Once again the accuracy drops quickly, this time with more variation and a shorter period before reaching 0 percent accuracy. LSTM significantly outperforms the other models for longer sequences.

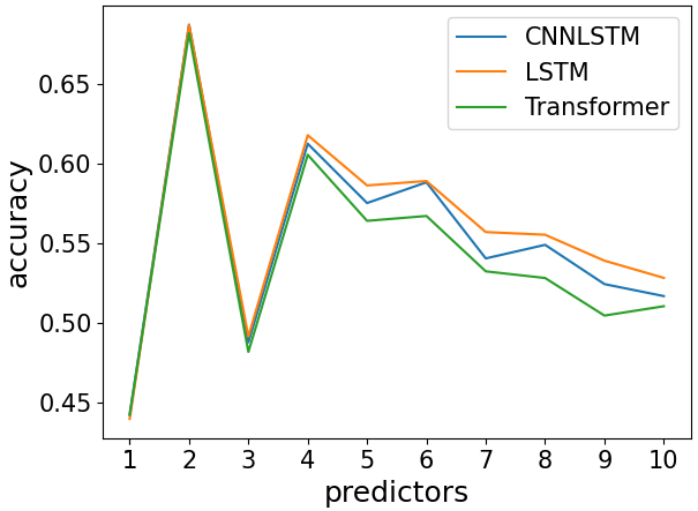

Figure 4.5 Next-action prediction performance as a function of the number of predictors on which the next action prediction is based.

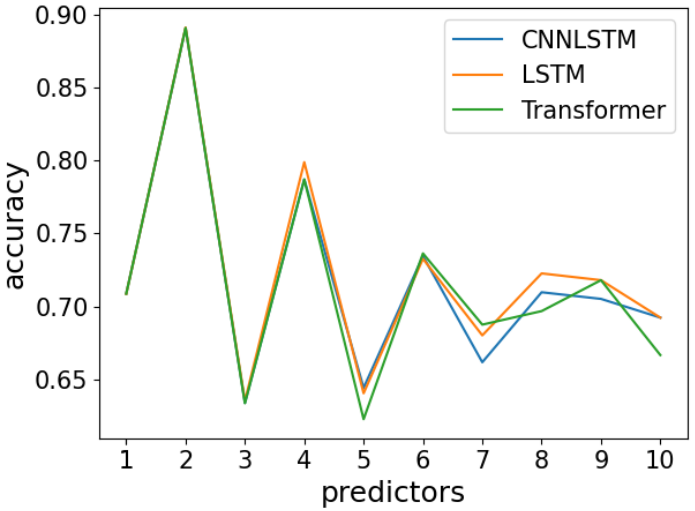

Figure 4.6 Next-action prediction performance as a function of the number of predictors on which the next action prediction is based, evaluated on topology-processed data.

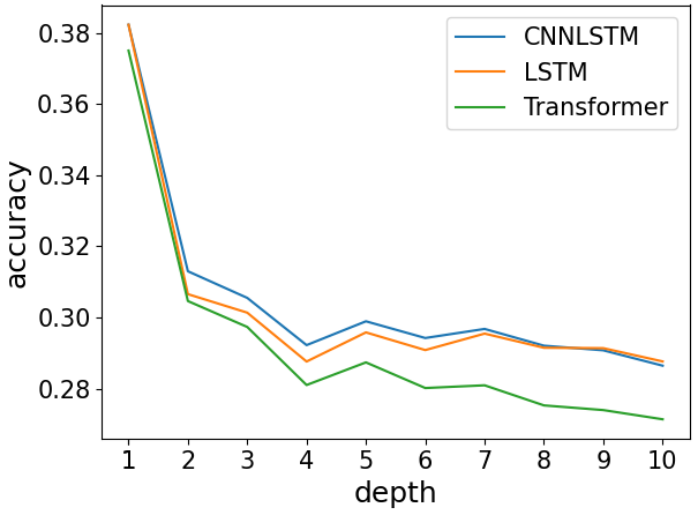

Figure 4.7 Last-action prediction performance as a function of label depth, i.e. the distance between predictors and label.

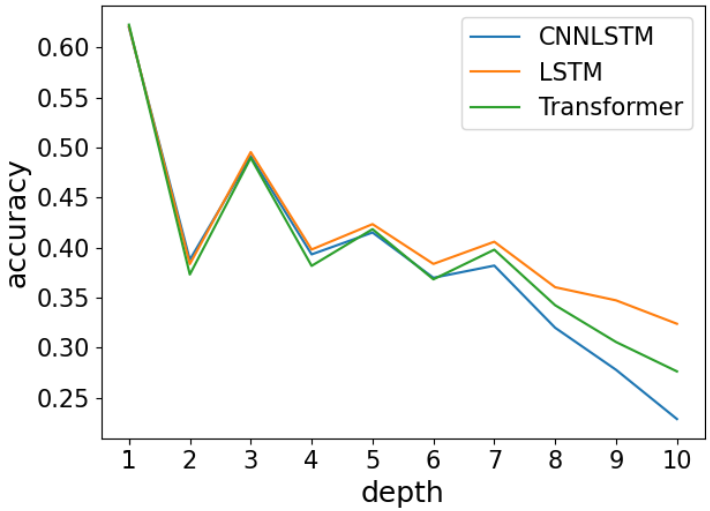

Figure 4.8 Last-action prediction performance as a function of label depth, i.e. the distance between predictors and label, evaluated on topology-processed data.

5

Discussion

5.1 Baseline

The overall performance of the baseline is worse than that of the ML models. This is understandable since the baseline is hard coded to follow the most common occurrences seen in the data. Additionally, the baseline only has the context of one classifier token to make a prediction and does not employ the knowledge of all previous tokens. Furthermore, no advanced back-propagation algorithm is employed to adjust the errors made by the model.

An interesting observation is that the baseline had better accuracy when producing its top-3 most likely predictions compared to the rest of the ML models when evaluated on the web topology-split dataset. This is most likely because the ML models are trained to maximize accuracy for a single prediction, while the baseline makes agnostic predictions without weights based on its probability vector.

5.2 LSTM

The LSTM model showed promising results across the different datasets. The best performance could be found when evaluated on the topology-split data with next-action predictions. Since the sequences of web actions follow specific branches in the web topology there are fewer actions to choose from, which leads to higher accuracy. Another reason for the increased accuracy is the fact that much uncertainty is removed when training the model to only make predictions along one branch without trying to make predictions of navigation between branches.

The best performance is also tied to the next-action prediction model variants. Naturally, it is more difficult to make predictions further ahead since more uncertainty is introduced, and thus next-action datasets score a higher accuracy than their last-action counterparts.

5.3 CNN-LSTM

The CNN-LSTM hybrid model was experimented with to make improvements to the LSTM model with the strengths of a CNN model. The CNN-LSTM hybrid outperformed the other models on last-action prediction using the non-topology split dataset, however only with a slight margin. The model performed similarly to the LSTM model with its overall best result achieved on the topology-split data for next-action prediction.

The main point of adding a CNN layer to the LSTM architecture was to explore if the strength of feature extraction that the CNN architecture provides could be combined with the powerful long-term dependency capturing that the LSTM architecture possesses. One reason why the CNN-LSTM architecture does not improve on the LSTM architecture could come from the lack of complex features present in the data.

5.4 Transformer

The transformer model has risen in popularity in recent years and is a great tool for solving NLP tasks. Naturally, since the task of predicting user behavior is treated as an NLP problem in this thesis, the transformer model was employed. However, this model achieved no notable improvements over the other two models.

The transformer model does well in complex problems involving a large number of dependencies. The processed data from the WUM produced a vocabulary size of 116 unique tokens. This is a very small number compared to how many individual tokens a vocabulary has in traditional NLP tasks. This is reflected in the results gathered from the transformer model which performed similarly to both the LSTM and CNN-LSTM models.

5.5 Performance

When looking at the performance across the different models it is notable that even though there are some slight differences in the results, the LSTM, CNN-LSTM, and transformer models perform similarly. The best overall accuracy was achieved using the LSTM model with the topology split dataset for next-action prediction using the top-3 predictions. The topology split dataset outperformed the normal dataset across all the models. This can be explained by the fact that when the different sequences were split based on the web topology, a lot of uncertainty was removed.

Furthermore, a jagged pattern can be observed at the start of most depth performance graphs. This can be attributed to a small number of typical use cases that occur with high frequency in the customer support tool. As the sequences get longer,

they diverge, but at short sequence lengths, certain specific sequences occur often enough to skew the results.

The performance of the different models starts to vary when the number of predictors in the next-action prediction and the depth for the last-action prediction increases. The LSTM architecture shows better results for longer sequences which is where it gets its advantage in accuracy over the other two architectures. When the number of predictors and depth gets too large all models start to perform unreliably. This comes from the lack of long sequences in the data. Limiting the length of training sequences with insufficient data could improve model performance. The downside of this is that the model would not be able to make predictions on input sequences exceeding the length limit.

Another performance improvement could come from the selection of data from specific offices, as certain departments handle specific issues. However, the information about the source of the data was limited to the country of origin, and doing additional training based on the office of origin would have been out of the scope of this thesis.

5.6 Insights

When working with the data coming from the customer support tool we realized that the complexity of the data might be lacking for the approach of ML. This can be seen in the results where the different models, although outperforming the baseline, still do not make great improvements over the baseline predictor. Additionally, the different model architectures perform very similar to each other.

A significant increase in accuracy was achieved from splitting sequences according to the web topology. A lot of uncertainty in the prediction of user web navigation came from when the user chose to visit another branch of the web tree. By predicting user navigation along the same branch a higher accuracy could be achieved. A downside of splitting the sequencers was that the average sequence length decreased to 2.85. Because of this, the complexity of an already limited web page was further decreased, leading to the different ML models performing even more similarly. This also decreases the accuracy of last-action predictions with a higher depth value, since depth is tied to sequence length.

The model variants trained to predict the next action in a sequence could be used for a dynamic UI where the user can more easily choose the next action. Last-action prediction is only useful if the user can skip some actions and achieve the same result on the web page. Meaning, that if the navigation of the web page does not contribute to the user's goal, the last-action prediction could be used to take the user to the desired destination instantly.

6

Conclusion & Future Work

Looking back at the goal of this thesis presented in Section 1.5, it can be stated that the goal of predicting future actions of web navigation and actions was achieved. The goal of achieving an accuracy higher than 70% on next-action prediction was also reached. An NLP approach was employed on the web log where each log entry was treated as a word in a sentence. This approach was applied to three different ML models and evaluated using the accuracy metric. The models were trained and evaluated on four different datasets, based on whether the data had been split on web topology, and if next-action or last-action n-grams were produced. The topology split dataset produced better results in accuracy, while at the same time bringing the different ML models closer in performance. Last-action predictions had overall lower accuracy than next-action predictions but allowed for predictions further into the future. Overall, the models performed very similarly, each having a dataset on which they outperformed the rest by a small margin.

It can be noted both from the data and the experiments that the complexity of the task of predicting the next action in a user web session for this particular web tool was lacking. The average web session length for the non-split sessions was 4.7 actions and the average session length for the topology split datasets was 2.85 actions per session. Additionally, the number of unique actions, or dictionary length, was determined to be 116. Because of this, we concluded that the lack of improvement between the different model architectures comes from the lack of complexity in the data.

Based on the results of the different ML models and the baseline it can be said that the ML approach in predicting user navigation and behavior on the web is promising. However, further experiments on larger websites and web tools could give better insights into how well the different models perform compared to each other.

6.1 Future Work

A potential benefit that could come from the prediction model is the pre-loading of certain web resources to decrease loading time, as the model could predict necessary resources needed ahead of time. Furthermore, a side effect of the proposed method is valuable statistics for user experience designers aiming to create a user-friendly interface. Thus, the method could be used to evaluate the effectiveness of different usability decisions. Future work could focus on applying the method proposed in this thesis to these alternative areas.

However, the main future goal emerging from the results of this thesis is to use the predictions that the different trained models are capable of producing to improve the efficiency of the customer support staff. To do this, the user's navigational actions need to be input to the model and the output from the model should in some way change the UI to help with faster navigation. A version of this would be to display the predicted next action to the user in a more discoverable fashion. With an implementation in place, A/B testing could be performed to evaluate the effectiveness of the approach, and to further improve models and processing techniques.

Bibliography

[1] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[2] Amazon. *What is a neural network?* www.aws.amazon.com/what-is/neural-network/. [Accessed 2023-10-12].

[3] L.-F. Bouchard. *What is self-supervised learning? | will machines ever be able to learn like humans?* www.medium.com/what-is-artificial-intelligence/what-is-self-supervised-learning-will-machines-be-able-to-learn-like-humans-d9160f40cdd1. [Accessed 2024-02-21].

[4] V. Chitraa and D. A. S. Davamani. "A survey on preprocessing methods for web usage data". *arXiv preprint arXiv:1004.1257* (2010).

[5] S. Cristina. *The attention mechanism from scratch*. www.machinelearningmastery.com/the-attention-mechanism-from-scratch/. [Accessed 2024-02-21].

[6] Crypto1. *How does the gradient descent algorithm work in machine learning?* www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/. [Accessed 2023-12-18].

[7] GeeksforGeeks. *What is web usage mining?* www.geeksforgeeks.org/what-is-web-usage-mining/. [Accessed 2023-12-14].

[8] GeeksforGeeks. *Wintroduction to convolution neural network*. www.geeksforgeeks.org/introduction-convolution-neural-network. [Accessed 2023-12-07].

[9] Google. *Word embeddings*. www.tensorflow.org/text/guide/word_embeddings. [Accessed 2023-11-17].

[10] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural computation* **9**:8 (1997), pp. 1735–1780.

[11] IBM. *What are convolutional neural networks?* www.ibm.com/topics/convolutional-neural-networks. [Accessed 2023-12-07].

[12] IBM. *What are neural networks?* www.ibm.com/topics/neural-networks. [Accessed 2023-10-12].

[13] IBM. *What are recurrent neural networks?* www.ibm.com/topics/recurrent-neural-networks. [Accessed 2023-10-12].

[14] IBM. *What is deep learning?* www.ibm.com/topics/deep-learning. [Accessed 2023-11-30].

[15] IBM. *What is machine learning?* www.ibm.com/topics/machine-learning. [Accessed 2023-10-11].

[16] IBM. *What is nlp?* www.ibm.com/topics/natural-language-processing. [Accessed 2024-02-21].

[17] A. Jung. *Machine Learning. The Basics.* Machine Learning: Foundations, Methodologies, and Applications. Springer Nature Singapore, 2022. ISBN: 9789811681929. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=cat02271a&AN=atoz.ebs30578841e&site=eds-live&scope=site.

[18] K. E. Koech. *Cross-entropy loss function.* https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e. [Accessed 2023-11-30].

[19] L. Liu, J. Liu, and J. Han. *Multi-head or single-head? an empirical comparison for transformer training*. 2021. arXiv: 2106.09650 [cs.CL].

[20] Manu. *A simple overview of rnn, lstm and attention mechanism.* www.medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07b. [Accessed 2024-02-21].

[21] G. Neelima and S. Rodda. "An overview on web usage mining". In: *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2*. Springer. 2015, pp. 647–655.

[22] A. Pai. *What is tokenization in nlp? here's all you need to know.* www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp. [Accessed 2023-11-24].

[23] K. B. Patel and A. Patel. "Process of web usage mining to find interesting patterns from web usage data". *International Journal of Computer Applications & Technology* **3**:1 (2012), pp. 144–148.

[24] M. Rahman. *Different ways to combine cnn and lstm networks for time series classification tasks.* www.medium.com/@mijanr/different-ways-to-combine-cnn-and-lstm-networks-for-time-series-classification-tasks-b03fc37e91b6. [Accessed 2023-12-06].

[25] C. Ruchini. *Introduction to word embeddings.* www.medium.com/analytics-vidhya/introduction-to-word-embeddings-c2ba135dce2f. [Accessed 2023-11-08].

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. “Attention is all you need”. In: I. Guyon et al. (Eds.). *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
February 2024

Document Number
TFRT-6224

Author(s)
Patric Balan
Gustav Jönemo

Supervisor
Albin Olsson, IKEA IT AB, Sweden
Johan Eker, Dept. of Automatic Control, Lund University, Sweden
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Predicting Navigational Patterns in Web Applications using Machine Learning Techniques

Abstract

In large corporations, customer support is a costly service, and an area of constant optimization. Solutions to increase efficiency and decrease bottlenecks are constantly needed. One such bottleneck is support tool proficiency in a customer support organization where many different tools are used, and a potential solution is to let the tool guide the user as it is being used. This thesis explores the use of machine learning to make predictions on user behaviour, based on user web log entries, to simplify the use of a customer support tool for unfamiliar users.

From 1.8 million log entries, four different datasets are created, based on two different data processing principles. On these, three machine learning models are trained, namely an LSTM, a transformer, and a hybrid CNN-LSTM model. These are then compared to a naive baseline model and evaluated on overall accuracy, top-three accuracy, and accuracy based on sequence length.

The results show that all trained models perform better than the baseline model, but not significantly for certain datasets. The trained models also perform very similarly on all datasets, but for long sequences, LSTM generally outperforms the others, reaching an overall accuracy of 75 percent for its best dataset. The respective accuracy for the baseline model is 72 percent. The close results between models can mainly be attributed to the low complexity of the tool from which the log entries originated and the few features they contain.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-51

Recipient's notes

Security classification

http://www.control.lth.se/publications/