CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Sequence classification applied to user log data

An approach to identify characteristics of user sessions in a music streaming service

Master's thesis in Computer Science – algorithms, languages and logic

Sofia Edström, Josefin Ondrus

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHEBURG
Gothenburg, Sweden 2017

Master's thesis 2017:12

Sequence classification applied to user log data

An approach to identify characteristics of user sessions in a music streaming service

Sofia Edström
Josefin Ondrus

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2017

Sequence classification applied to user log data
An approach to identify characteristics of user sessions in a music streaming service
Sofia Edström, Josefin Ondrus

Supervisors: Oscar Carlsson, Spotify AB.
Richard Johansson, Department of Computer Science and Engineering.
Examiner: Alexander Schliep, Department of Computer Science and Engineering.

Master's Thesis 2017:12
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Sequence classification applied to user log data
An approach to identify characteristics of user sessions in a music streaming service
Sofia Edström, Josefin Ondrus

Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Applying machine learning techniques to sequential user log data can provide insights about users that can guide companies towards making decisions that improve user experience. Recurrent neural networks have been proven to work well in combination with sequential data and recent research suggests that incorporating residual connections in recurrent structures outperforms standard recurrent structures. In this thesis, we show that residual recurrent neural networks can be applied to user log data from a complex domain in order to identify regularities in user behavior. To our knowledge, no research have been conducted with these model structures in domains other than text and image classification. A proof of concept is implemented in collaboration with Spotify where this approach is used to identify how users behave when they save music in the music streaming service. By conducting experiments with different models, we show that models with increased input complexity slightly outperform models with lower input complexity in the artificial classification task defined in this thesis. We also show that results from a more complex model can be analyzed and provide valuable insights. However, we conclude that the approach is ineffective and needs more developement in order to become sufficient.

Keywords: sequence classification, user log data, residual learning, recurrent neural networks.

Acknowledgements

This thesis was conducted for and made possible by Spotify AB. We would like to thank our company advisor Oscar Carlsson for his support throughout the thesis work and all the colleagues at Spotify that have helped us whenever we needed it. Furthermore, we would like to thank and express our gratitude to our supervisor Richard Johansson at Chalmers University of Technology and our opponents Christian and Simon. This thesis would not have been possible to conduct without their help.

Sofia Edström, Josefin Ondrus, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1

Introduction

As the field of computer science has evolved, so have the methods of user research [1]. Today, cloud services and the declining cost of hardware enable storing of and computing with large data sets [2]. The combination of large amounts of user data, cheap computing hardware and cloud services, have driven the trend towards the use of machine learning techniques in a variety of settings. Applying machine learning techniques to user log data is one possibility that can provide insights about users, which can guide companies towards making decisions that improve user experience [1].

Examples of research on the subject of user log data in combination with machine learning are adaptive unix interfaces, resource use prediction and e-commerce product recommendations [3, 4, 5]. In previous research, the techniques used and experiments conducted vary depending on the context and the definition of the problem. Most studies within the area aim to predict future user actions or are focused on giving suggestions based on similar users. However, there is a limited amount of research that analyzes application usage, and few documented data sets built upon user logs.

This thesis investigates whether machine learning can be applied to user logs to identify how users behave when they save music in a music streaming service. This is done by applying state-of-the-art sequence classification methods [6, 7], on a data set consisting of sequences constructed by user logs generated by users of the Spotify music streaming service.

Recurrent neural networks have been demonstrated to work well on sequential data. Furthermore, recent experiments suggest that incorporating residual connections in recurrent structures outperforms standard recurrent structures [8]. In this thesis, experiments are conducted on five different architectures of recurrent neural networks, where four of them have residual connections incorporated. These experiments are conducted in a supervised setting, using a data set consisting of user logs.

The purpose of this thesis is to further explore the possibilities of using machine learning in combination with user log data to provide insights about application usage. By using the approach described in this thesis, we extend the work presented by Wang and Tian on residual connections in recurrent structures [6]. By presenting performance measurements of models trained on user log data, this work aims to contribute to research in the field of sequential learning. Moreover, we contribute to Spotify's user research by investigating the results of the experiments and by analyzing the data's effect on model performance.

1.1 Aim

The aim of the thesis is to investigate whether residual recurrent neural networks are suitable to use for user logs to gain insight into application usage. This is done by implementing a proof of concept using sampled real-world anonymized log data, with a feature space consisting of of both numerical and categorical features, gathered from users of the Spotify application. The goal of the thesis is to identify how users behave in the Spotify application when they save music. In this thesis we aim to answer two questions:

- Do the models perform well on the types of data collected from the user logs?
- Can we find indicative features and session characteristics to describe application usage?

1.2 Scope

A recently conducted study of the effectiveness of recurrent neural networks in sequence classification is "Recurrent Residual Learning for Sequence Classification" by Wang and Tian [6]. The study presents experiments conducted with six different recurrent architectures in different classification tasks. Because of the well documented experimental setup and recent progress in the field presented, the paper was used as a base for the experiments in this thesis. Moreover, the work by Wang and Tian is extended by performing further experiments with recurrent architectures on a newly introduced artificial classification task described in section 3. The experiments are conducted in a supervised setting using a data set constructed of user logs.

There is limited research on user log data in the field of machine learning. A lot of the research within the area investigates models to predict future user actions given a part of a sequence. Frequently used methods in previously conducted research include Hidden Markov Models [4], Boosting Decision Trees [9], and recurrent neural networks [10]. With inspiration from earlier research in combination with our academic expertise and interest, we limit this thesis to only cover recurrent neural networks.

As stated in section 1.1, the aim of this thesis is to investigate if residual recurrent neural networks for sequence classification are suitable to use in combination with a data set constructed from user logs. Since Spotify expressed interest in this research and since we could not locate any public collections of user logs, the data used in the experiments is based entirely on user logs from Spotify. Moreover, the objective of the classification task is defined and scoped in collaboration with Spotify.

The problem of this thesis, described in section 3, is formulated as a classification task of entire sequences. However, as the class label is known through some of

the events, although censored, it should be considered a constructed classification problem. This implies that the classification in itself is not important, but only serve the purpose of exploring the existence of underlying patterns in the data and measuring the performance of model architectures.

1.3 Related research

To our knowledge there exists no previous research tackling the problem of classifying entire sequences of data based on user logs similar to those described in section 3.1. These user log sequences differ from other types of sequences studied in the field of sequence classification in that they consist of time-dependent data with a feature space of mixed feature types, i.e. data with both numerical and categorical features. Other user log sequences studied typically involve data with a feature space of only a subset of those feature types.

Previous state-of-the-art research in sequence classification has not considered mixed feature types [7, 11]. Moreover, there exists research on suitable models (mainly different kinds of recurrent neural networks) for predicting future elements of sequences of such data, but not for classifying entire sequences of it [9, 10].

1.3.1 Applications on similar data

Some of the research on anomaly detection and business process monitoring applies event sequence prediction to data of mixed feature types [9, 10].Xie and Coggeshall solve two sequence prediction problems related to patient hospital visit records by using a stochastic gradient boosting decision tree [9], and Tax et al. have solved several prediction problems related to business process instances using recurrent neural networks [10]. However, the above mentioned research has aimed to predict future events in a sequence, not to classify entire event sequences.

1.3.2 Sequence classification

Recurrent neural networks (RNNs), and specifically variants of RNNs with Long Short-Term Memory (LSTMs), are effective tools for solving problems with sequential data [7]. Wang and Tian explore the possibility of incorporating residual connections in RNNs and propose different models in line with this idea: a Residual RNN, Skip-Connected LSTM:s, and a Hybrid Residual LSTM [6]. They compare their results to state-of-the-art models for sequence classification and conduct several experiments supporting the effectiveness of the new models. However, the classification is applied to one dimensional, atomic representations of textual and image based data, not to sequences of mixed feature types.

1.4 Thesis Outline

The thesis is outlined as follows. In section 2, a detailed account of the theoretical background on which our approach is based is presented. In section 3, our approach for investigating application usage by classifying sequences constructed by user logs is described in detail. Further, a presentation of the user log data and the methods used for evaluation and result analysis is given. Section 4 describes and discusses the experiments and the results. A comparison of the performance of the different model architectures is presented. The section also includes results from the different methods of analysis and discusses key findings. In section 5, conclusions based on the work in this thesis are drawn and suggestions for future work are given.

2
Background

In this section, a detailed description of the theoretical background on which our approach is based is given. Some fundamental concepts of artificial neural networks are described, followed by more detailed descriptions of recurrent structures and residual learning. In order to provide context to the data used in the experiments, the concept of the Spotify application and some background on the user log data is presented.

2.1 Spotify user data

Spotify is a music streaming service available on a wide range of platforms in 60 markets across the world[1]. The service offers features such as searching for, browsing, and playing music, videos and podcasts. It allows its users to manage their own content library by, for example, saving music, creating playlists and adding music to them. Spotify has more than a hundred million monthly active users, which together account for a large amount of user sessions per day. The amount of user sessions, derived from event log data generated by user actions performed in the application, ensures that Spotify can provide enough data to build a sufficiently large data set.

A user session can be defined as a sequence of consecutive events derived from a single user. Such sessions are created by exploring a set of event logs, mapping the events in them to users, and chronologically ordering the events into a sequence. An event log is a record over either user actions performed in the application or events happening as a result of such actions. An event log can consist of information that is relevant for a specific feature or is interesting to inspect for a specific reason, for example information about a song that was streamed or when in-app navigation occurred. Which kinds of events are included in a user session depend on which set of event logs the session is derived from.

2.2 Classification models

Classification is the problem of learning the underlying function that maps some feature set x to some class label y. A classifier is a model that has learned this function by adjusting the parameters of the function and that can map new instances with the feature set x to a class label. There are many different types of classification

[1] www.spotify.com

models. Examples are decision trees, rule-based classifiers, support vector machines and artificial neural networks. A type of artificial neural networks that handle sequences of input particularly well is recurrent neural networks (RNNs). Because of this characteristic, RNNs were used in the experiments presented in this thesis, and are consequently described in this section.

2.2.1 Artificial neural networks

Given an input vector $x \in \mathcal{R}^n$, some weight parameters W, and activation functions ϕ, a neural network produces an output vector $\hat{y} \in \mathcal{R}^M$. When using a neural network as a classifier, it is common that M is the finite number of classes defined for the set of instances and that each element in $\hat{y}$ represents a different class. The $softmax$ function is commonly used as activation function in the final layer of the network. The function transforms the input from the previous layer to a probability distribution that produces an output vector $\hat{y}$. The output vector contains probabilities that sum up to one. The instance then usually gets assigned the class label of the highest probability. An example of a simple feed forward neural network with two hidden layers can be seen in Figure 2.1.

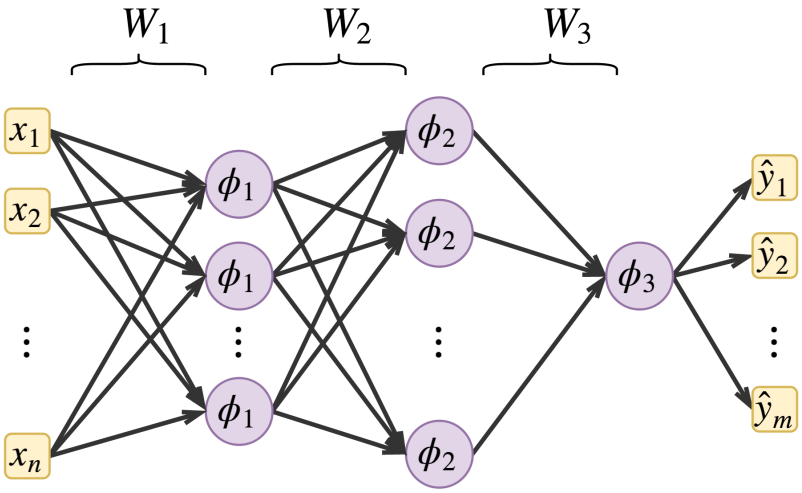


Figure 2.1: A feed forward artificial neural network with two hidden layers. The squares represent input and output vectors, the circles are neurons with activation function ϕ and W_l is the weight parameters of layer l.

The process of learning the weight parameters W of a neural network is often divided into two categories: supervised and unsupervised learning. If the labels of the instances are unknown, the neural network has to be trained in a unsupervised setting. On the other hand, if the labels of the instances are known, supervised learning can be applied and is often preferred.

2.2.1.1 Backpropagation and gradient descent

When training a neural network classifier in a supervised setting, the objective is to adjust the parameters of the network such that the difference between the predicted labels $\hat{y}$ and the true labels y is as small as possible. A common training algorithm is the backpropagation algorithm, which can be described as an iterative process of finding parameters that minimizes the difference between the labels [12, 13]. In combination with an optimization algorithm, for example gradient descent, the parameters are adjusted to find the minimum error of the network. A function defining this error is called the loss function, $\mathcal{L}$, which is a function of $\hat{y}$ and y. The objective of backpropagation is to find parameters that minimizes the function $\mathcal{L}(\hat{y}, y)$.

There are different loss functions, and their suitability varies depending on the activation function of the output layer. For $softmax$, the cross entropy loss function is commonly used. The cross entropy loss can be seen in Equation 2.1, where N is the number of instances and M is the number of different classes in the training data.

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{N} \sum_{n \in N} \sum_{m \in M} y_{n,m} \log \hat{y}_{n,m} \tag{2.1}$$

The general approach is to update the weights W_l in layer l, of the network by making corrections ΔW_l, based on a learning rate η and the gradient g of $\mathcal{L}$. The weights in layer $l + 1$ is updated accordingly:

$$W_{l+1} = W_l + \Delta W_l, \tag{2.2}$$

where

$$\Delta W_l = -\eta g_l. \tag{2.3}$$

The gradients of the weight parameters are calculated by propagating backwards through the network, from the output layer to the weights of the input layer, using the chain rule of differentiation. The algorithm then minimizes the error by searching the function space of the loss function $\mathcal{L}$, following its gradient by updating the weights, to find a local minimum.

2.2.2 Recurrent neural networks

The fundamental idea of RNNs is that they share weights through multiple time steps, that allow the information to persist from the input throughout the network [7]. RNNs are often illustrated as neural networks with loops, which when unfolded, resemble a chain-like structure as can be seen in Figure 2.2. This architecture makes them suitable for sequential input, as each loop corresponds to one element of the sequence being processed through the network. During the propagation a hidden state vector and an output vector are generated at each iteration. Each iteration can be seen as a time step t, where the hidden state at time step t is produced by the previous time step and input x_t. In sequence classification tasks, only the last output is of interest, creating a so called "many-to-one" structure as can be seen in figure 2.2.

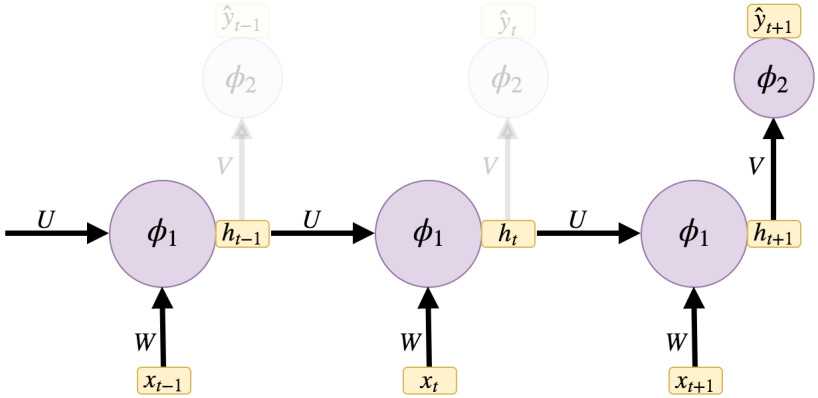


Figure 2.2: An unfolded representation of a recurrent neural network with a "many-to-one" structure. The rectangles represent input and output vectors, the circles are neurons with activation function ϕ and U, W and V are the shared weight parameters.

In Figure 2.2, the input sequence is represented by $x = \{\cdots, x_{t-1}, x_t, x_{t+1}\}$, where $x_i \in \mathcal{R}^n$. The input weights and the inter-state weights are denoted by W and U, respectively, and are shared across time steps. The weight parameters V between the hidden state and the output state are also shared across steps. However, in this "many-to-one" structure the output is only calculated when the entire sequence has been propagated.

The hidden state $h_t \in \mathcal{R}^m$ and the output $\hat{y}_t$ at time step t is produced by the previous hidden state and the input at time step t as follows:

$$\begin{aligned} h_t &= \phi_1(W \cdot x_t + b_W + U \cdot h_{t-1} + b_U) \text{ and} \\ \hat{y}_t &= \phi_2(V \cdot h_t + b_V). \end{aligned} \tag{2.4}$$

The hidden state activation function ϕ_1 is often a nonlinear function such as $tanh$, ϕ_2 often $softmax$, and b is the respective layer's corresponding bias vector. In subsequent sections the bias vectors b will be omitted from the equations for legibility.

2.2.2.1 Backpropagation through time

The backpropagation in a recurrent neural network largely resembles that in a regular neural network. As in regular backpropagation, the parameters are adjusted in the search for a local minimum of the loss function $\mathcal{L}$. The difference is that the gradients need to be calculated for the weights at each time step, since the hidden weigh parameters W and U are shared across time steps, and later time steps depend on earlier ones.

Looking at an unfolded recurrent network, the weights can be updated as in 2.2 by recursively computing the gradients for the different layers. The difference is that when calculating ΔW and ΔU, the total sum of the gradients from all the time steps is used instead of just the local ones.

When optimizing deep structures, like recurrent neural networks, using gradient based learning algorithms with a fixed parameter value for the learning rate η can be problematic and result in exploding or vanishing gradients. In 1994, this was discussed by Bengio et al. in [14], where they show that there is a tradeoff between efficient learning and remembering long time dependencies.

Different methods have been introduced to tackle the problem of vanishing gradients in deep structures. One way is to use alternative gradient based learning algorithms as opposed to the standard gradient descent. One example is AdaDelta, which adjusts the learning rate based on previous gradients [15]. To tackle the problem of exploding gradients it is common to use gradient clipping [16].

Another approach that has been shown as effective when learning long term dependencies is to help the network to remember information from previous time steps by incorporating connections between the time steps. Such a structure is called the Long Short-Term Memory, where so called gates are incorporated, alongside the hidden layer of the network. Another is to use identity mappings called residual connection, between the layers.

2.2.3 Long Short-Term Memory

A popular variant of an RNN is the Long Short-Term Memory (LSTM), which was first introduced by Hochreiter and Schmidhuber in 1997 [17]. In LSTMs, the architecture and concept of shared weights from the standard recurrent neural network still persist. However, a memory cell state is incorporated in the structure, enabling the network to learn long term dependencies, which have been proven to be effective when working with longer sequences.

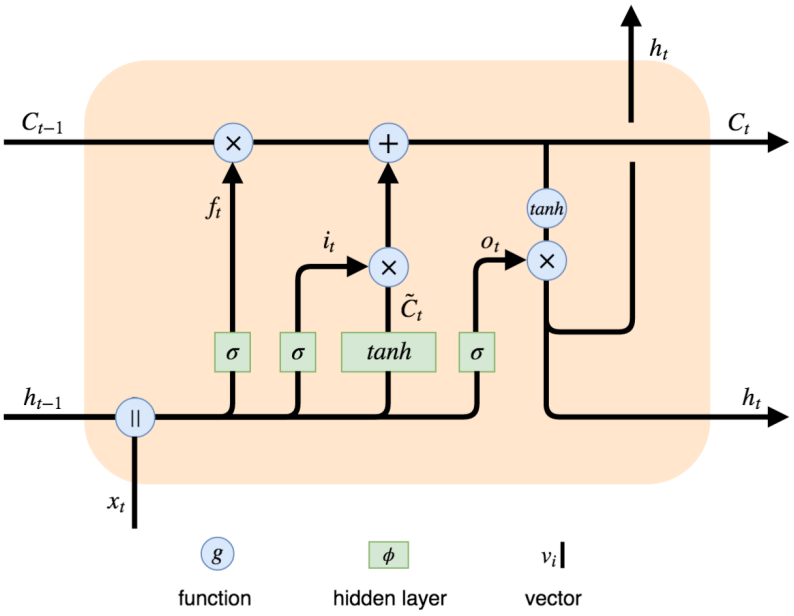


Figure 2.3: The structure of an LSTM unit.

In an LSTM structure the hidden state is not only produced by the weight parameters and the previous hidden state, but also by a cell state, $C_t \in \mathcal{R}^m$. The LSTM regulates which information to keep in long term memory, and which information to forget, through the cell state. Three internal gate signals are affecting the cell state and the hidden state. These signals are called the input, output and forget gates, and are denoted as i_t, o_t and f_t, respectively 2.5. The expression $h_t, C_t = LSTM(x_t, h_{t-1}, C_{t-1})$ will be used when referring to h_t and C_t produced by a LSTM unit.

$$\begin{aligned} f_t &= \sigma(W_f \cdot x_t + U_f \cdot h_{t-1}) \\ i_t &= \sigma(W_i \cdot x_t + U_i \cdot h_{t-1}) \\ \tilde{C}_t &= tanh(W_C \cdot x_t + U_C \cdot h_{t-1}) \\ o_t &= \sigma(W_o \cdot x_t + U_o \cdot h_{t-1}) \\ C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t \\ h_t &= o_t \times tanh(C_t). \end{aligned} \tag{2.5}$$

In 2.5, $\cdot$ is the dot product, $\times$ is the element-wise product and $\sigma(x) = 1/(1 + exp(-x))$ is the sigmoid function. Here there are different weight parameters W and U for each hidden layer in the LSTM unit. Note that there are no weight parameters after the internal hidden layers.

2.2.4 Residual neural networks

In 2015, He et al. introduced identity (skip) connections called residual connections, to enhance information flow in deep architectures for image recognition [8]. These residual connections between layers are meant to ensure direct propagation of signals, which in turn prevents the gradient from vanishing during training.

Similar to the cell state C_t of an LSTM unit, a residual network persists the information from previous layers l by passing this information to a future layer $l + 1$ with a more clear signal. A residual unit does this pass without modifying the signal at all, generating h_{l+1} with the following transformation:

$$h_{l+1} = \phi_{l+1}(W_l \cdot h_l) + h_l, \tag{2.6}$$

where h_l is the direct output from the previous hidden layer, W_l are the weight parameters between the previous layer and the current layer $l+1$ and ϕ_{l+1} is the activation function. Note that the weight parameters W_l are not shared and can vary between the different layers. Also note that in 2.6 there are no inter-state weights, since the function describes a residual layer in a feed forward neural network, not in a recurrent neural network.

The use of residual connections was proved to be an effective method in the deep architectures used by He et al.[8]. Also Liao and Poggio discuss the concept of time-invariant neural networks with shared weights in combination with residual connections and found them to work well [18]. Furthermore, Wang and Tian explores the possibility of time-variant recurrent neural networks with residual connections [6]. They conduct experiments with multiple architectures that indicate that it is a successful concept.

2.2.5 Residual recurrent neural networks

The concept of residual recurrent neural networks is to add residual connections between the different time steps in a recurrent neural network structure. The propagation through the residual connections in a residual RNN differs somewhat from the one in a feed forward residual neural network. In the latter case, the output from the previous layer is propagated through the residual connections to the next layer. In the former case, it is the hidden state h_{t-1} from the recurrent unit that gets propagated directly to the next state. This forces a direct transfer between every two consecutive time steps.

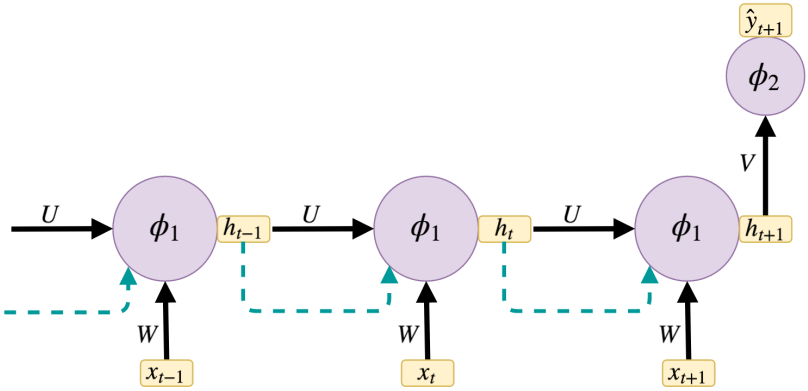


Figure 2.4: An unfolded representation of a residual recurrent neural network. A dashed line represents a residual signal from the previous time step. The rectangles represent input and output vectors, the circles are neurons with activation function ϕ and U, W and V are the shared weight parameters.

In Figure 2.4, the residual connections in an unfolded recurrent neural network are visualized as dashed green lines between the different time steps. The input weights, inter-state weights and output weights are denoted as W, U, and V, respectively, and are shared across time steps. The t^{th} element x_t in the sequence, together with the hidden state signal from the previous time step, is used as input. The activation function of the hidden layer, often the *tanh* function, is denoted by ϕ_1. The activation function in the output layer, which in classification tasks often is the $softmax$ function, is denoted by ϕ_2. The function produces the output $\hat{y}_t$.

In a residual recurrent neural network, the signal from the previous hidden state is passed to the next time step, affecting h_t. Similar to the residual function 2.6, the previous hidden state h_{t-1} is added to the output of the current hidden unit:

$$\begin{aligned} h_t &= \phi_1(W \cdot x_t + U \cdot h_{t-1}) + h_{t-1} \\ \hat{y}_t &= \phi_2(V \cdot h_t). \end{aligned} \tag{2.7}$$

2.3 Evaluation

When evaluating a classifier, it is of interest to measure the generalization capability of the model. This is done in order to get an understanding of how well it represents the underlying class distributions, i.e. how well it would classify new instances with unknown class. There are some common metrics and techniques which can be used to estimate how well a model performs on new instances and compare the performance of different models.

2.3.1 Metrics

Common metrics to use when evaluating supervised classification tasks are precision and recall [19]. The metrics can be used to determine how well a binary classifier performs on a specific task, given the true class labels and the predicted class labels for a set of instances. The metrics use the different outcomes that are listed in the confusion matrix in Table 2.1. When combining precision and recall, one can calculate accuracy, F-score, or shift the weights of the two metrics to let one affect the result more than the other, if needed. The metrics are defined as:

$$\text{precision} = \frac{TP}{TP + FP}$$

and

$$\text{recall} = \frac{TP}{TP + FN},$$

where precision is the proportion of true positives among all positive instances, and recall is the proportion of true positives among all instances classified as positive. precision and recall are defined relatively to a specific class and it is commonly the positive class.

Accuracy

The accuracy is the proportion of correctly classified instances and does not require the precision and recall metrics. However, it does require the total amount of correct predictions. It measures how well the model performs in the classification task, and is often used as a statistical measurement for different types of classifiers. It is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

F-score

The F_β score is used to calculate the weighted average of precision and recall. It generates a measured score between 1 and 0, where 1 is a perfect score. For $\beta = 1$, precision and recall are weighted equally, and the F score is equal to the harmonic mean. For positive real values where $\beta < 1$, precision is weighted higher than recall. In these scenarios, a β-value of 0.5 is the most commonly used.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{2.9}$$

Table 2.1: Confusion matrix of the possible outcomes of a binary classifier.

		assigned class	
		positive	negative
actual class	positive	true positive (TP)	false negative (FN)
	negative	false positive (FN)	true negative (TN)

2.3.2 The hold out method

The purpose of a classifier is to find an underlying distribution, given a set of samples from that distribution. To successfully find that distribution, it is important that the classifier avoids to fit to noise or variance of the samples. If the classifier does not avoid this, it has lost its generality, which results in unwanted behavior and erroneously made predictions when classifying new instances.

The behavior when a model fits to the training data's noise or variance is referred to as overfitting. There are some common techniques to avoid this unwanted behavior and ensure a proper estimate for model performance. One such technique is called the hold out method. It is based on withholding a subset of the samples during the training procedures, and later use these samples to test the classifier. This procedure provides an unbiased estimate of how well the classifier performs on new instances of unknown class, which can help achieve model generality.

The hold out method is not only used for testing the classifier, but also for detecting when to stop the training in order to prevent overfitting. The method includes dividing the set of sample instances into tree subsets used for training, validation and testing the classifier.

The training set is used during model training. It is usually the largest set, to ensure that the model has enough sample instances to succeed in its generalization. The validation set is also used during model training, but not to adjust model parameters based on the predictions of it. Instead, the validation set is used to get an estimate of model generality during the training procedure.

By validating the model performance in intervals, it is possible to estimate when the model is overfitting. An indication of overfitting is that validation accuracy decreases while training accuracy increases. When the validation accuracy decreases, the training is stopped and the weight parameters are retrieved from the iteration when the validation accuracy was the highest. The test set is used to get a final estimate of the model performance, and should never be seen by the model before the evaluation is made.

3
Approach

This section contains a comprehensive explanation of the approach used to answer the two questions defined in section 1.1. First, we investigate if there exist underlying patterns and regularities in the sessions where users save music. This is done by classifying sequences of user logs and measuring the performance of the different models on these sequences. Next, we search for indicative features and session characteristics which we interpret in order to gain insights about application usage. The overall method consisted of four main steps:

1. To define and model a user session in such a way that it includes all relevant information provided by different user logs.
2. To construct a suitable data set for supervised learning based on these session sequences.
3. To apply appropriate models for sequence classification on this data set to determine if the results indicate the existence of underlying patterns in the users' sessions.
4. To search for indicative features and session characteristics to describe application usage by investigating model input sensitivity and feature relevance.

3.1 Modeling a user session

In the Spotify application, a user session is defined as the session of activity from when user activity is detected, until no activity takes place for a 30 minutes period. During that time, a user performs a number of actions, referred to as active events. Some of these active events can result in that the application perform a number of passive events. Both passive and active events indicate that the application is being used. An example of an active event that results in a passive event is a a press on the play button of an album. The pressed button results in a series of passive events during which the application is playing content from that album. During this time, the user may or may not perform active events. Hence, a user session is a series of events that can be both sequential and parallel. Even though events can occur in parallel, a user session is modeled as a sequence of ordered events that can be both active and passive, $s = [s_1, s_2, \ldots, s_T]$. In the sequence s, event s_i starts before s_{i+1}, and s_T is the last event that occurred before a 30 minute period of no recorded events.

There are four different categories of events: *stream*, *navigation*, *playlist-addition* and *library-modification*. The event categories originate from six different event logs. A *stream* event occurs every time audio content is played, a *navigation* event occurs when a user performs in-app navigation, a *playlist-addition* event happens when content is added to a playlist, and *library-modification* occurs when content is added to or removed from a user's library. An example session sequence k, where a user performs some actions and listens to four songs can be seen below.

$$s^{(k)} = [navigation, navigation, stream, stream, stream, playlist\text{-}addition, stream]$$

Each event contains some general information about the event, such as the duration of the event and a timestamp of when the event took place. Each event also has a set of specific details which varies depending on which category of event it is. Examples of event category specific details are which page was navigated to in a navigation, or which type of content that was played in a stream. To make use of all information available in the event logs, both the general and event category specific data is encapsulated in the feature vector. This is achieved by modeling each event category as a vector with the dimension of the number of features for that event category. In table 3.1, the dimensions of the four different event categories are listed.

Table 3.1: Number of event category specific features

Event category	Feature quantity
stream	13
navigation	1
playlist-addition	3
library-modification	2

3.2 Data set construction

A data set suitable for supervised learning was constructed from a set of sampled user sessions, which are described in 3.1. This data set consists of a set of instances, where each instance k has the form of a tuple $(x^{(k)}, y^{(k)})$. Here, $x^{(k)}$ is a sequence of ordered elements x_i, all of equal dimension. The element dimension was determined by the total number of distinct features across the sequences. Each sequence has a corresponding class label $y^{(k)}$, which was determined based on the goal of the proof of concept. That is, identifying how users behave in the Spotify application when they save music.

The first step was to assign labels to each session, s, based on events that directly indicated that a user saved music. The existence of save events in a session determined if it would be classified as a session during which a user saved music. If $s^{(k)}$ contained at least one such event, that session was classified as "positive". If $s^{(k)}$ did not contain any of the events, it would be classified as "negative". All sessions were classified using this procedure, making it possible to later transform the classifications to class labels y.

Before constructing the instance sequences x, all events that directly indicated that music was saved were removed from the sessions. This since we wanted to search for underlying patterns in the user behavior before and after the users saved music, not search for the actual events directly. As the class labels were based exclusively on these events, this should be considered an artificial classification problem. However, the belief was that if a classifier performs well (significantly better than chance), there are underlying patterns in the user sessions. These patterns can then be attempted to be interpreted in order to gain additional insights about application usage.

With the event categories e as basis, a combined feature space for each event x_i was defined by the concatenation of all features. The result was a total of 20 different features, including the event category e itself, and three other general features. Note that information about event category specific features was only available in events of that specific category. Since the data provided was real-life data, some sequences had missing or corrupt values, which needed to be handled. Furthermore, many of the features were categorical, and required some preprocessing to function as input to a neural network. The numerical features also required some preprocessing, such as scaling the values to an appropriate range.

3.2.1 Preprocessing and feature representation

The data preprocessing steps consisted of imputing missing values, feature scaling and one-hot encoding categorical features. The preprocessed result of the sequences was matrices, where each row was represented by an event x_i. These events were in turn represented by a feature vector consisting of one-hot encoded categorical features and scaled numerical features. An overview of the different types of features and their respective quantity in the feature space are displayed in table 3.2.

Table 3.2: Quantities of different feature types in the data set.

Feature type	Quantity
event category ∈ *{stream, navigation, library-modification}*	1
categorical	9
time tracking	1
numerical	9

Since the sessions were constructed from real-world event log data, some values were

initially missing. Less than half of the features had missing values, where the proportion of such values was between 0.2% and 12.1%, depending on feature. One of the reasons as to why values were corrupt or missing is that sometimes not all information gets successfully or correctly recorded in an event. Missing and corrupt values were handled differently depending on the feature type. In the sessions, missing values of numerical features generally indicated zero, and were therefore imputed with "0" in the feature vectors. The time tracking feature, TIMESTAMP was guaranteed to always be present, and was therefore not affected by this basic approach. Corrupt or missing categorical values were replaced by the category "unknown". The categorical features sometimes had outlier values, for example search strings or identifiers. The outliers were reinterpreted as new categories, for example *search* or *identifier*.

Feature scaling was performed by normalizing the numerical features. That is, all numerical values were rescaled in the range $[0, 1]$, where the smallest value was replace by zero and the largest value was replace by one. This because it is suggested that neural networks converge faster when the data is distributed in the vicinity of the output. All the numerical values except the TIMESTAMP feature were normalized feature-wise across all sequences to preserve the relationships between the values. To preserve the relative time between the events in a sequence, the TIMESTAMP feature was normalized sequence-wise.

The values of categorical features needed to be numerical in order to be represented in the input feature vector of a neural network. The categorical values are nominal, and were therefore converted into a one-hot encoded vector per feature. A one-hot encoded vector is a one-of-r representation of a value of a nominal feature that can take on r unique values. For every event, one of the r elements in the vector is set to "1" to indicate the active value, while the rest of the elements are set to zero to indicate passive values. A one-hot encoded example of the categorical feature EVENT CATEGORY is presented in equation 3.1.

$$\begin{aligned} \textit{stream} &\mapsto [1, 0, 0] \\ \textit{navigation} &\mapsto [0, 1, 0] \\ \textit{library-modification} &\mapsto [0, 0, 1] \end{aligned} \tag{3.1}$$

The final representation of a session sequence $x^{(k)}$ was on the form of a 20-dimensional feature space with a total of 227 dimensions. This feature space contained all the information gathered from the original sessions. Each sequence $x^{(k)}$ had a corresponding one-dimensional label $y^{(k)}$, representing the class of sequence k. The classifications previously assigned to the sessions were transferred to the corresponding sequence such that $y^{(k)} = 1$ if the class of sequence k was positive and $y^{(k)} = 0$ if the class was negative. An example sequence matrix can be seen in Figure 3.1.

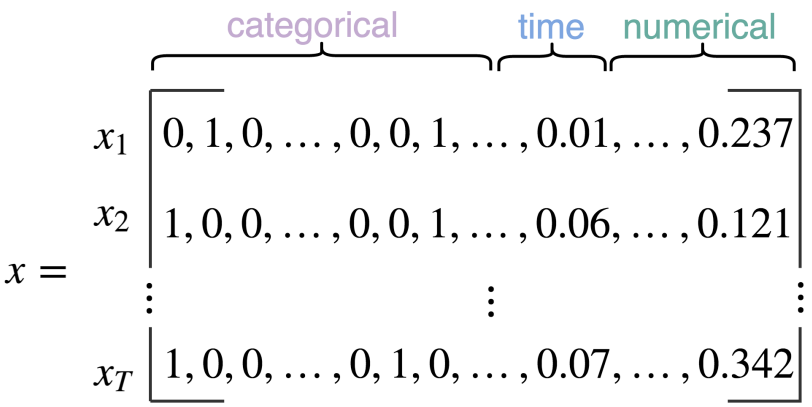


Figure 3.1: An example sequence x of length T represented by T feature vectors, where x_i corresponds to event i.

3.3 Sequence classification

Related research mentioned in section 1.3 has shown that recurrent neural networks are suitable tools not only for sequence classification in general, but that they also work well on distributions that originate from log data similar to the one described in section 3.1. However, some of the weaknesses of simple recurrent structures appear when applying them on longer sequences, where they are struggling with long time dependencies and problems as vanishing or exploding gradients can appear during training.

Different strategies to handle these limitations of simple recurrent neural networks are presented in section 2, such as incorporating connections between time steps, in order to remind the network of previous information. Examples of such structures are the Long Short-Term Memory network (LSTM) and residual recurrent neural networks (RRN). In [6], Wang and Tian take advantage of the two approaches and introduce LSTM structures with residual connections. Based on their results, it seems that these succeed well in the classification task, hence it is the approach chosen for solving the classification task of user log sessions.

It is not clear from previous experimental results, which of the model types are best suited for solving the classification problem of the user sessions. Previous experiments conducted with these model architectures have been with either categorical or numerical feature spaces. However, to take advantage of all the information provided in the user logs, the sequences constructed from the log sessions have a feature space of mixed categorical and numerical features. Hence, we perform experiments with all three architectures that take advantage of both LSTM units and the residual connections.

Two of the architectures are LSTMs with residual connections between the hidden states, referred to as Skip Connected LSTMs (SC-LSTM). What differentiate them is the number of hidden states the residual connections are skipping. In SC-

LSTM-I, the residual signal is passed directly to the next time step. In the other version, SC-LSTM-P, the residual signal skips $p > 1$ time steps before being added to the hidden state. The respective definitions of the two models are presented in Equation 3.2. Their respective abstract architectures are illustrated in Figure 3.3. Note that in SC-LSTM-P, the residual connections are not present in every time step, but rather in the time steps $1, 1+p, 1+2p, \ldots, 1+\lfloor\frac{T-p-1}{p}\rfloor p$, for $p > 0$.

$$h_{t+p} = \tanh(c_{t+p}) \times o_{t+p} + h_t \tag{3.2}$$

The third version of the architectures that take advantage of both LSTM units and residual connections, is a hybrid of the two called the Hybrid residual LSTM (HRL). HRL is two networks that run in parallel throughout an entire sequence, whose independent signals are then combined before propagating them to the classification layer. This combined signal is obtained as the mean of the two hidden states h_T^{RRN} and h_T^{LSTM}, which can be seen in Equation 3.3. An abstract visualization of the hybrid is presented in 3.3.

$$h_t^{HRL} = \frac{1}{2}(h_T^{RRN} + h_T^{LSTM}) \tag{3.3}$$

Two additional models, which do not combine LSTM units and residual connections, are used in the experiments as well. This is to get an understanding of the effects of the combination of the two techniques. The first is a "vanilla" LSTM, which is described in detail in section 2.2.3. In the "vanilla" LSTM, both the cell state C_t and the hidden state h_t is the input to the LSTM unit in time step $t+1$. The second model is an RRN, described in section 2.2.5, where h_t is the sum of the output from the hidden unit at time step t and the previous hidden state h_{t-1}. Equation 2.7 provides a definition of h_t in an RRN.

A total of five models are used in the experiments of this thesis. The properties that apply to all models are the recurrent structure and the final layer. All models except HRL only have one single hidden unit. The general architecture of the models can be seen in 3.2. Abstract illustrations of each of the models can be seen in Figure 3.3.

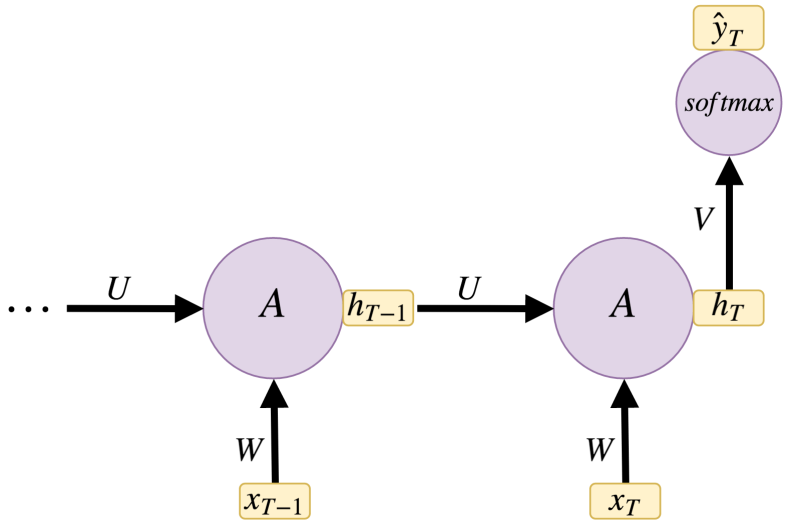


Figure 3.2: The general architecture which was applied to all classifiers used to solve the task. The hidden unit A varies depending on the model type.

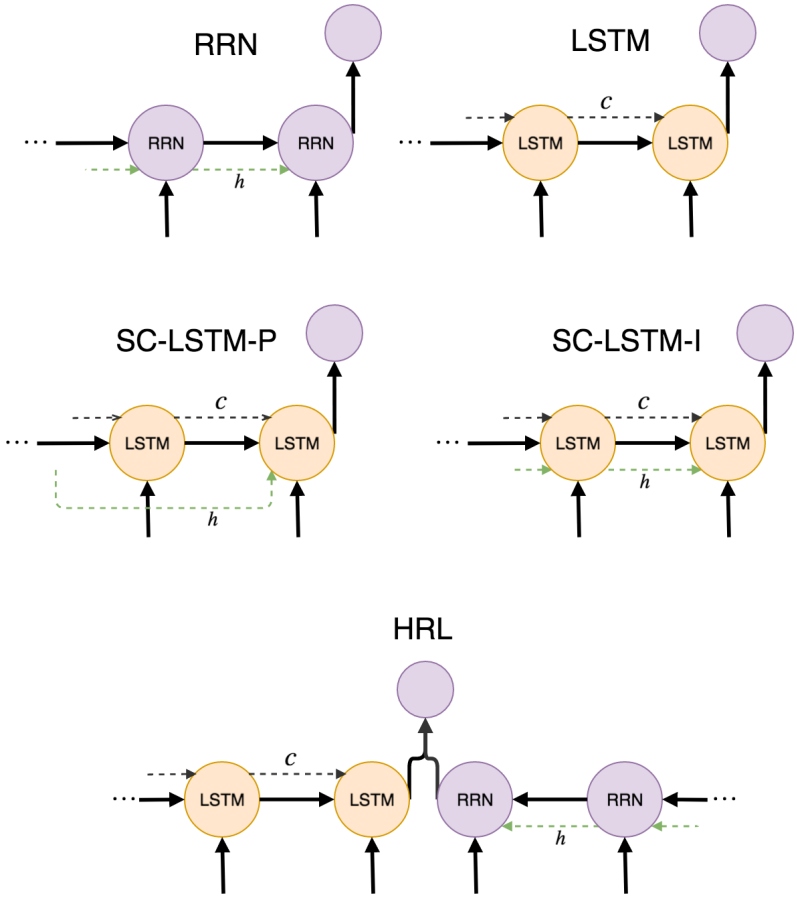


Figure 3.3: The architectures of the models.

3.4 Experiment design and evaluation

To answer the two questions in section 1.1, two different types of experiments were carried out. The first consisted of evaluating performance for different classifiers on a number of data set versions with varying feature space. In the second experiment, model input sensitivity and feature relevance was measured by iteratively removing features from the full feature space.

3.4.1 Model performance experiment

To answer question one in section 1.1, the five different model architectures defined in section 3.3 were trained and evaluated on four different versions of the data set. The approach was to increase the feature space and measure model performance on more descriptive versions of the data set. The four versions were designed to give an understanding of how well the different architectures performed with different kinds of features.

The first version, designed to work as a baseline, consisted of simple atomic events. Forming a one-dimensional feature space, the EVENT CATEGORY feature was used to construct the sequences, with no information about time or other possibly indicative information. With this structure, the input vectors was much like the text and pixel based data used in previous experiments on these model architectures [6].

The remaining three versions were defined as follows. The second version of the data set was defined by all the categorical features available, including the feature from the atomic version. The feature space of version three was defined by the categorical features together with a time tracking feature. The fourth version version included the full feature space, that is, categorical, time tracking and numerical features.

If a model outperformed its baseline on a more descriptive version of the data, it was reported as capable of of processing different types of information. For example, if the results improved when the feature TIMESTAMP was included as a feature in $x^{(k)}$, it was concluded that the model could process time-dependent multidimensional categorical data. Moreover, if the results continued to improve when numerical features were incorporated, this was interpreted as an indication that the model could also handle time-dependent, multidimensional sequences with both categorical and numerical data. Consequently, the classification results on the multiple data set versions are indications as to whether the models performed well on sequential data constructed by user log events.

To evaluate the classification performance of a model, the test accuracy was measured according to equation 2.8. The accuracy was determined by using the hold-out method and calculating the percentage of correctly classified instances in a test set containing only sequences not previously seen by the model. The number of instances in the test set was a fourth of the total number of instances in the entire

data set.

3.4.2 Feature relevance experiment

The second suite of experiments was conducted in order to answer the second question in section 1.1, and to investigate model input sensitivity. This time, the feature space was decreased, while the classification accuracy was measured. During these experiments, the model architecture that seemed to be performing best during the previous experiments was used. Its classification accuracy on the full feature space version of the data set was used as baseline.

One classifier was trained and tested on 19 different versions of the data set. Each version corresponded to the removal of one single feature, resulting in a feature space of 19 dimensions. The class label prediction results were then compared to that of the full feature space. Further result analysis, described in section 3.5.2, was conducted with those features whose removal caused the most significant change in classification results.

3.5 Result analysis

A number of different tests were conducted to analyze the results from the experiments, in order to provide answers to the questions in section 1.1. This included analyzing the effect of different feature types on model performance, hypothesis testing to determine significant differences between the models, and detecting indicative values of features by comparing distributions of feature values between sessions with different predicted class labels.

3.5.1 Model performance analysis

To analyze if any of the model architectures in the experiments performed statistically significantly better than the others, the non-parametric Friedman's Test was used [20]. The model architecture was the independent variable of the test, the different data set versions was the samples, and the classification accuracy of a model on a data set version was the dependent variable. A visual representation of the statistical test setup is presented in table 3.3.

		model				
		LSTM	RRN	SC-LSTM-1	SC-LSTM-P	HRL
data set version	Atomic	acc	acc	acc	acc	acc
	ECD	acc	acc	acc	acc	acc
	TECD	acc	acc	acc	acc	acc
	PCD	acc	acc	acc	acc	acc

Table 3.3: Setup for Friedman's Test.

To analyze whether the classification results across the different data set versions are statistically significantly different, the non-parametric statistical McNemar's Test was conducted. 2x2 contingency tables were constructed, one for each possible combination of two versions of the data set. The values a, b, c and d in table 3.4 are the comparison of the predicted labels from the models on two different data set versions, where a is the number of instances labeled correctly in both versions, b is the number of instances labeled erroneously in version 1 but correctly in version 2, c is the number of instances labeled correctly in version 1 but erroneously in version 2 and d is the number of instances labeled erroneously in both versions.

		data set version 1	
		correct	wrong
data set version 2	correct	a	b
	wrong	c	d

Table 3.4: Contingency table of predicted labels on two different data set versions.

By performing a hypothesis test for the data set version pairs, a statistically significant difference was either determined or not. The null and alternative hypotheses are formulated in 3.4. The null hypothesis h_0 states that the probability that a session label is predicted erroneously in the first version is equal to the probability that the session is being predicted erroneously in the second version. If the null hypothesis was rejected, the difference in class label predictions made on the two versions was determined as statistically significant.

$$\begin{aligned} h_0 &: p_b = p_c \\ h_1 &: p_b \neq p_c \end{aligned} \tag{3.4}$$

Error analysis of the models was performed by creating a confusion matrix reporting true positives, true negatives, false positives and true negatives, and then inspecting the session sequences in the different categories. In order to get an indication of where the models fail to predict the correct label, both the false negative and false positive sessions were compared against both categories of correctly predicted sessions. The comparison was made by inspecting the average values of a number of variables for the erroneously predicted sessions, and testing if they are similar or different to the ones of the correctly predicted sessions. Among the variables covered are session length (number of events), duration, number of occurrences of a specific event category, and the number of addition events.

In [6], Wang and Tian experienced that some of their models did not perform well on short sequences, but that the performance improved with increased sequence length. To investigate the effect of the sequence length, an experiment was conducted with a version of the data set where the events in a session were replaced by a constant value. The classification result was then compared to those on the other versions of the data set. In addition, the relationship between classification accuracy and the sequence length was investigated by partitioning the test set into batches based on length, and then reporting the accuracy as a function of sequence length.

3.5.2 Feature relevance analysis

In order to find which features are important for the model to successfully predict session labels, experiments with different features removed from the full feature space data set were conducted, as described in section 3.4.2. McNemar's Test was used to determine if a removed feature caused a significant change in label prediction results. This was done by comparing the labels predicted by the model when a feature was removed to those predicted on the full feature space. The test was repeated for all removed features. The results were used as a measure of both model input sensitivity and to understand which features are relevant when classifying the sessions.

If a removed feature caused a statistically significant change in classification results, further analysis of feature values by inspecting the classification outcome in the full feature space case was performed. If the feature was numerical, the distributions of feature values of positively predicted sessions were compared to those of negatively predicted sessions. If the feature was categorical, the occurrences of the feature values of positively predicted sessions were recorded and compared to those of the negatively predicted sessions. The comparisons were summarized and plotted, and were used to analyze which values of the relevant features that were associated with, and indicative for, the respective classes.

4
Experiments

In this section, the conducted experiments, along with the data set versions used, are described. Moreover, model parameters and experimental setup are reported. Finally, the results and analysis from the different experiments are presented.

4.1 Data

In all experiments, the same training, validation and test partitioning of the data set was used. This ensured that the model was trained, validated and tested using the same instances independent of experiment or data set version. How the sequences were modeled is described in Section 3.1, and a detailed explanation of how the data set was constructed is given in Section 3.2. The general properties of the data set are presented in Table 4.1.

Table 4.1: Properties of the data set.

Total number of instances:	31774
Training partition:	0.5625
Validation partition:	0.1875
Test partition:	0.25
Instances labeled positive:	15887
Instances labeled negative:	15887

4.1.1 Data set versions for evaluating model performance

The first suite of experiments, described in more detail in section 4.2, was conducted with four different representations of the data. The versions are described in Table 4.2, and are listed below:

1. Atomic version (Atomic), where an event is represented by one categorical feature.
2. Multidimensional categorical version (CF), where an event is represented by all categorical features.
3. Time-dependent multidimensional categorical version (TCF), where data set version 2 is extended to include the TIMESTAMP feature. The feature represents the relative time between the first event and the current event in each sequence.
4. Full feature space version (FF), where an event is represented by all the features available, including the numerical ones.

Table 4.2: Different representations of the data set used in the experiments.

Version	Abbr.	Feature kinds	Number of features	Input vector size
1	Atomic	categorical	1	3
2	CF	categorical	10	217
3	TCF	categorical, time	11	218
4	FF	categorical, time, numerical	20	227

4.2 Experimental setup

The implemented training module, and all the classifiers experimented with, were written in Python. The source code used as a basis, implemented by Wang and Tian [6], is publicly available at the publishing service of the University of Illinois[1]. Several changes were made to the source code, where the most significant ones allowed us to use multidimensional input vectors. A Python library called Theano[2] was used in combination with two NVIDIA Tesla K80 graphical processing units, for parallel tensor calculations.

The cross entropy loss, defined in equation 2.1, was used as loss function, since it is commonly recommended when using $softmax$ as activation function in the last layer. We use AdaDelta [15], which is a gradient based optimization algorithm. It is used in combination with backpropagation through time. A gradient clipping value of 1.0 was applied to avoid exploding gradients [16].

All experiments were performed using equal settings. That is, all static parameter settings were kept constant during the training of all models, across all datasets. The hold out method with a validation frequency of 10 epochs was used. When no improvements in validation accuracy were recorded for hundred epochs, early stopping was performed. The number of hidden units was set to 600, which resulted in weight matrices U and W having dimensions of *input vector size* $\times$ 600. The sizes of the different input vectors can be seen in Table 4.2. The number of instances in the training, validation and test sets can be seen in Table 4.1.

Since neural networks are complex structures with many parameters to optimize, the training time usually is relatively long. Therefore a method called mini-batch gradient was used, which is known to speed up training without compromising the classification performance too much. A mini-batch size of 16 instances per mini-batch was applied. The training time varied between 2 and 4 minutes per epoch, depending on the hidden state dimension and type of model. The algorithm usually converged after 200 epochs, and then ran for another hundred epochs according to the early stopping procedure. The parameter and training settings were decided af-

[1] https://publish.illinois.edu/yirenwang/home/emnlp16source/
[2] http://deeplearning.net/software/theano

ter some initial training trials. The settings used were chosen because they enabled the network to converge with adequate results within a reasonable time frame.

4.3 Results

This section presents the results from the model performance experiments and feature relevance experiments. Key findings and the significance of results are demonstrated and commented on.

4.3.1 Model performance

To investigate the model architecture's effect on classification accuracy for different versions of the data set, we evaluated all combinations of models and data set versions. The results per model and data set version are presented in Table 4.3. In addition, a small test on model stability was conducted with the SC-LSTM-I model. The model was trained and evaluated a few times on the same data set version, which resulted in classification accuracy scores with a variance of 0.023%.

According to the outcome of the Friedman's Test, no model architecture performed significantly better than the others. However, the SC-LSTM-I model was initially observed to perform slightly better on some of the data set versions. This lead to the use of this model when measuring if there was a significant difference in model performance between the data set versions.

To test the difference in model performance on different data set versions, we compared all versions using McNemar's test and a Bonferroni corrected significance level of $\frac{0.05}{5} = 0.01$. This showed us that the label prediction outcome for the classifier when trained on the atomic (baseline) version of the data set was significantly worse than when trained on all other versions. However, no significant difference was measured between any of the other versions of the data set.

Table 4.3: Classification accuracy for the model experiments. Performances of each model are shown for each version of the data set.

Model	Atomic	CF	TCF	FF
LSTM	76.11%	80.55%	80.65%	81.03%
RRN	75.77%	81.03%	**80.87%**	81.07%
SC-LSTM-I	76.25%	**81.13%**	80.74%	**81.28%**
SC-LSTM-P	76.04%	80.38%	80.54%	80.97%
HRL	**76.36%**	80.92%	80.77%	80.54%

4.3.2 Feature relevance

In order to investigate model input sensitivity and feature relevance, a number of experiments, with one feature removed from the feature space at a time, were conducted. As mentioned in Section 4.3.1, the model used in these experiments was SC-LSTM-I, since it initially seemed to perform slightly better than the other models. The classification accuracy, precision, and recall were measured in all experiments. The measured values were then compared to the ones measured for the full feature version of the data set.

The resulting differences in classification performance for the removed features are presented in Table 4.4. The rightmost column shows if the removal of a feature caused a significant change in classification outcome compared to the outcome of the full feature space, based on the p-value. To point out statistical significance, McNemar's test was used in combination with a Bonferroni corrected significance level of $\frac{0.05}{20} = 0.0025$. The features with most impact on the classification outcome were the PAGE, MS_PLAYED and SHUFFLE features, with impact in decreasing order.

Table 4.4: Difference in classification performance for feature relevance experiments. The differences are calculated with respect to the performance of the SC-LSTM-I architecture trained with the FF data set version.

Removed Feature	Precision	Recall	Accuracy	p-value	significant
page	-3.08%	-1.75%	-2.63%	1.08e-12	Yes
ms_played	-2.83%	+2.12%	-1.08%	2.10e-04	Yes
shuffle	-0.79%	-1.49%	-1.06%	2.25e-04	Yes
source_start	-1.60%	+0.15%	-0.97%	0.002	Yes
ms_tot_est	-2.07%	+1.31%	-0.85%	0.003	No
n_seekfwd	+0.27%	-2.45%	-0.77%	0.006	No
n_stutter	-1.54%	+0.73%	-0.72%	0.009	No
media_type	-2.27%	+2.25%	-0.65%	0.014	No
bitrate	+0.14%	-1.92%	-0.64%	0.019	No
product	-1.30%	+0.78%	-0.54%	0.033	No
content_type	+0.45%	-2.13%	-0.53%	0.043	No
n_seekback	+1.10%	-3.19%	-0.55%	0.047	No
reason_start	-0.19%	+0.15%	-0.45%	0.116	No
n_tracks	-1.69%	+1.79%	-0.43%	0.127	No
timestamp	-1.31%	+1.21%	-0.39%	0.144	No
operation	-1.09%	+0.99%	-0.33%	0.209	No
duration	+1.40%	-3.04%	-0.31%	0.241	No
reason_end	-0.17%	-0.08%	-0.14%	0.597	No
ms_latency	-0.73%	+0.93%	-0.11%	0.653	No

4.4 Analysis and discussion

In this section the results from Section 4.3 are analyzed and discussed in more detail. Topics explored are the lack of performance difference between model architectures, the effect of an increased feature space on performance, how the models handle time-dependent mixed feature types, and the effect of sequence length on performance. Characteristics of values of the features that were determined relevant in Section 4.3.2 are presented along with possible interpretations of the cause of them.

4.4.1 Model performance

As can be seen in Table 4.3, all models performed roughly equal on the various data set versions. This indicates that the choice of recurrent model architecture was not of crucial importance for the classification task in this thesis. However, the lack of significant difference in classification results between models was surprising, since the results from Wang and Tian's experiments show greatly varying classification accuracy scores for the different architectures [6]. A possible explanation is that their experiments compare the performance of different model architectures on data sets that consist of sequences with long-term dependencies. They test if the new architectures they introduce handle such dependencies better than other state-of-the-art models, and point out significant improvements. The lack of significant difference between the models in our classification results could be explained by the lack of long-term dependencies. However, due to the scope of this thesis, we have not investigated weather the user logs in our data set demonstrate such long-term dependencies, hence we will not draw any conclusions regarding how well the different model architectures handle them.

Table 4.3 shows that there is an observable tendency towards increased classification accuracy when extending the data set with more features. This was particularly noticeable from the atomic version to the other versions of the data set. In addition, there was a small, but statistically significant, difference in label prediction outcome between the atomic version and the other data set versions. The increased classification accuracy, and change in label prediction outcome, show that the models perform well on a high dimensional feature space of categorical data. However, the models' prediction outcomes did not change significantly when expanding the data set feature space with timestamp and numerical features.

Despite the increased accuracy, we cannot fully state that the models perform well on time-dependent mixed feature types. This because no significant change in label prediction outcome was detected when adding timestamp and numerical features to the feature space. However, according to the results from the feature relevance experiment presented in Table 4.4, one of the numerical features caused a significant change in prediction outcome compared to that of the full feature space version. This contradicts the results of the model performance experiments. A possible explanation could be that the timestamp and numerical features might correlate with each other, or the other features, in a way that does not change the label prediction

outcome when they are added to the feature space. We also suspect that other feature scaling approaches than the one used in this thesis could increase the effect of some of the numerical features. Regarding the effect of the TIMESTAMP feature, a possible explanation could also be that it is not relevant for classification, since users likely express unequal levels of activity throughout sessions. However, the feature might have changed the outcome significantly if the task would have been to predict future events given subparts of user logs.

4.4.2 Sequence length

We investigated the impact of sequence length by training and evaluating a SC-LSTM-I model with a data set containing no other information than the number of events per session. The classification accuracy of the model was measured to 69.90%. The results can be seen in Figure 4.1. The model classification accuracy in the length experiment was significantly worse than that in the experiment with the atomic data set version. The latter resulted in a classification accuracy of 76.25%. These results show that the sequence length is relevant for the classifier, but not solely conclusive. In order to decrease the impact of sequence length on classification accuracy one approach could be to introduce multilayered hidden states. These additional hidden layers, which might enable them to capture other dependencies, could decrease the impact of sequence length by increasing the impact of other dependencies.

The experiments on the impact of the sequence length showed that length was important for the classification outcome. As can be seen in Figure 4.1, both precision and recall for the negative class decrease as sequence length increases. For the positive class, the trend is the opposite, i.e. both precision and recall increase with sequence length. This is not surprising, since true positive sessions in general had more events than true negative sessions, as can be seen in Figure 4.2. Based on the average sequence lengths we can observe that the sequences that are falsely classified as positive have a length more similar to the true positives than to the true negatives. The length differences are observed in the incorrectly classified negatives too, where the sequence lengths are more similar to the true negatives than to the positives.

4.4.3 Feature relevance

We investigated the impact of the different features by removing one feature from the full feature space at a time, to identify which were relevant. The results can be found in Table 4.4. They show that the most relevant features, i.e the features that changed the label prediction outcome significantly, were PAGE, MS_PLAYED, SHUFFLE and SOURCE_START.

We could observe additional interesting details from Table 4.4, such as a the feature MS_PLAYED being relevant despite its numerical feature type. We also notice that the TIMESTAMP feature seems unexpectedly irrelevant. Possible reasons are discussed in more detail in section 4.4.1. The fact that the classification accuracy

Figure 4.1: Precision and recall in proportion to session length ranges.

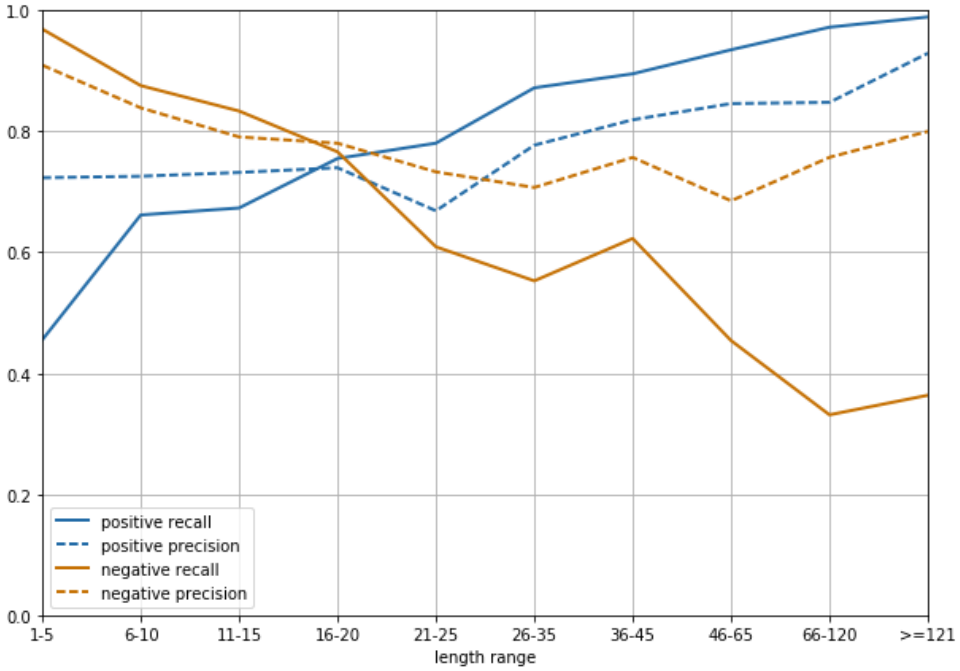
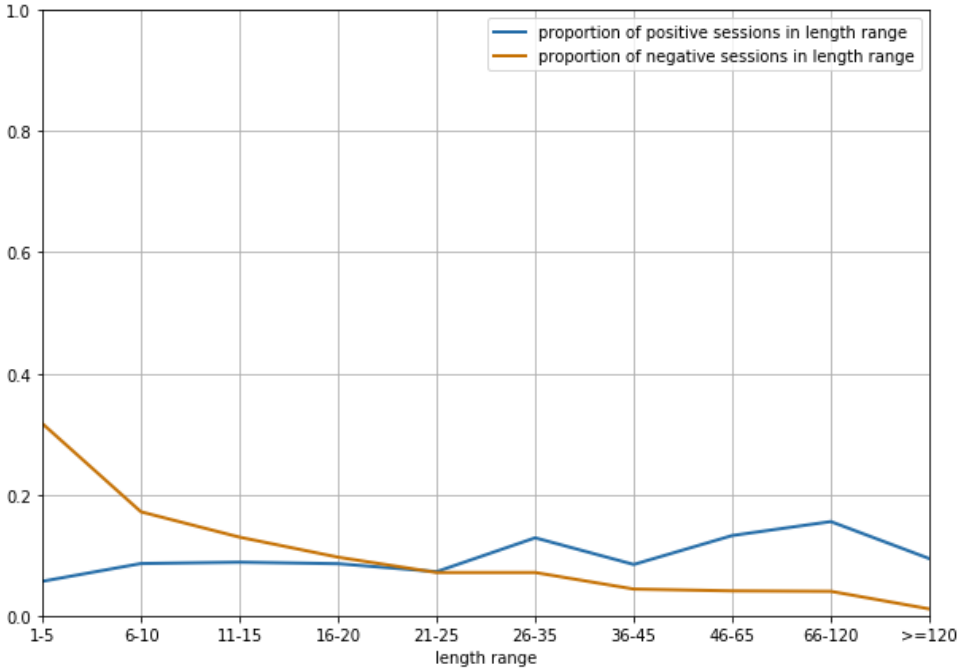


Figure 4.2: Proportion of positive and negative classes in proportion to session length ranges.



decreased in all feature removal experiments surprised us. The negative overall difference in accuracy was expected, since information was withheld. However we also expected that some model variance, in combination with the removal of possibly less relevant features, would result in some accuracy increase.

Further investigations of the features PAGE, MS_PLAYED, SHUFFLE and SOURCE_START were conducted. We analyzed the occurrence of different values, and distributions of values, to find session characteristics of the two classes.

PAGE

We studied the different values of the PAGE feature, and present the most common values with respect to the model's predictions. This was done by investigating the proportion of total amount of page views, and focusing on the pages that were exclusively in one of the predicted classes. In addition to this, we also looked at the pages where the proportion of the total amount of page views differed the most between the two predicted classes.

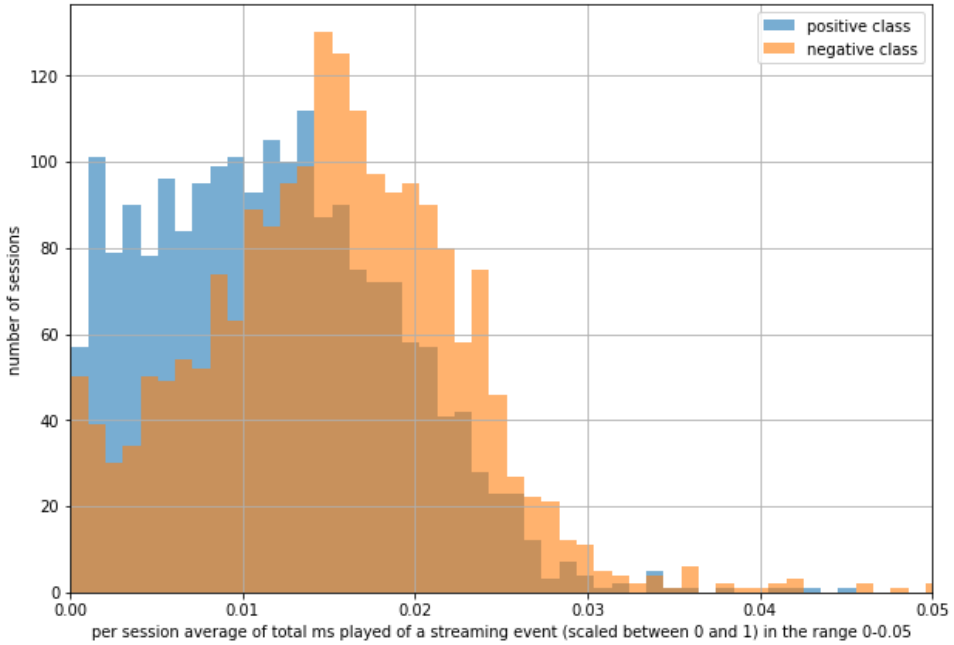
When examining the pages with the highest proportion of page views that occurred exclusively in one of the predicted classes, we found both expected and unexpected occurrences. It is clear that the page for adding songs to a playlist is the page with the strongest association to a single class, namely the positive class. This is not surprising, but we want to point out that there are sessions where this page is visited, but no songs are added. Hence, those sessions get incorrectly labeled. Overall, it seems that visiting different playlist pages is associated with the positive class. This is probably because it indicates that the user is actively browsing playlists and making changes to hir music library. We also want to point out that the more common case, when a user saves music to hir music library without adding it to a playlist, is not bound to a specific page as when saves are made to playlists. However, we can observe that certain pages, where it is possible to save music to the music library, are stronger associated to the positive class than other similar pages. A somewhat strange observation is that the majority of the pages associated with the negative class are pages concerning concerts. We are having a hard time understanding why, and leave further investigations to Spotify.

When examining the pages where the proportion of the total amount of page views differed the most between the two predicted classes, we found some additional associations. We noticed a tendency towards that sessions where users open the context menu are associated with the positive class. Similar to the playlists case mentioned, it is possible that this indicates that the users are actively using more rare features. Hence, they are overall more inclined to save music. We also observed that the connect/playback page is more associated with the negative class. We were not not surprised by this, since the page allows users to play music from another device than the one they are controlling the music from. This could indicate that users are listening through external speakers. Using external speakers increases the possibility of several people listening, leading to a likely decrease in application engagement due to social activity.

MS_PLAYED

Figure 4.3 shows that the negatively classified sessions generally had a larger average of ms played per streaming event than the sessions classified as positive, which agrees with the distributions of ms played per streaming event for the true classes. This could be an indication that sessions during which users are more selective of their music have more skipped songs than other sessions. Something else that can affect the value of MS_PLAYED feature is the actual length of the song played. However, it does not seem likely that sessions where users listen to shorter songs are more prone to include saves.

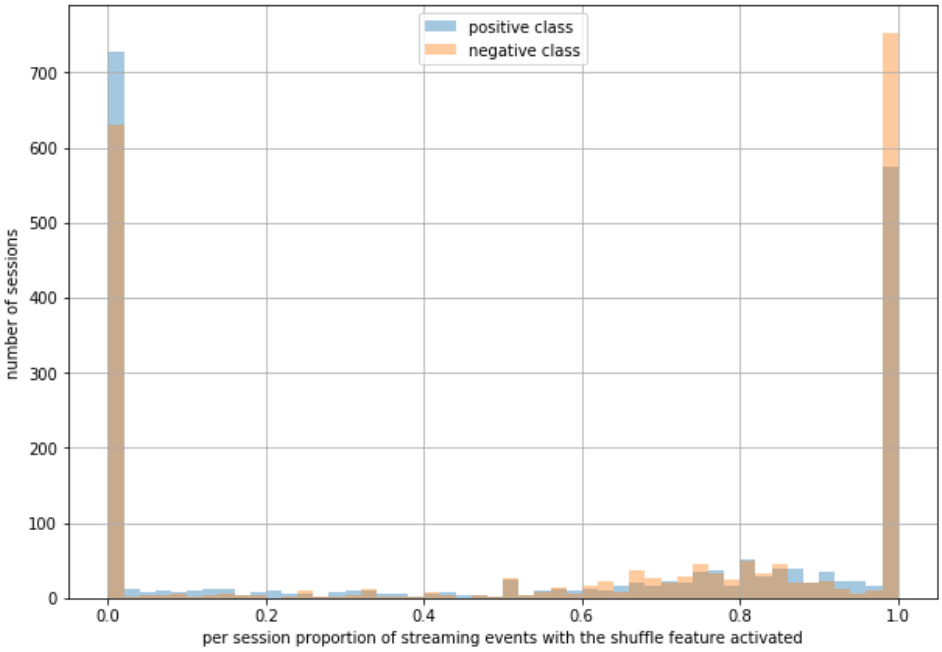
Figure 4.3: Average of ms_played per session in the two predicted classes. For the positive class, the mean was 0.01152 and the variance was 0.00006. For the negative class, the mean was 0.01552 and the variance was 0.00019.



SHUFFLE

In Figure 4.4 a tendency towards that the shuffle feature is activated more often for the negative sessions than for the positive sessions can be observed. This could perhaps be explained by the same reasoning as for the MS_PLAYED feature, i.e that a session in shuffle mode indicates that a user is less selective, and therefore less likely to save music. However, the difference between the predicted classes is not as noticeable for the SHUFFLE feature as it is for the MS_PLAYED feature.

Figure 4.4: Proportion of streaming events with shuffle activated per session in the two classes, 0 means that shuffle never was activated, 1 means it was always activated.



SOURCE_START

It is not that surprising that the SOURCE_START feature is relevant. The feature holds information about from where in the application a user started playing a song. As with the PAGE feature, the SOURCE_START feature indicates where in the application a user navigates, which we previously have concluded as relevant information. By observing the differences of occurring values of SOURCE_START between the two predicted classes, we have noticed some regularities. First, it is clear that sessions where recommended music is played are overrepresented in the positively predicted class. An explanation could be that a tendency to listen to recommended music indicates a will to explore music outside the library. Moreover, sessions classified as negative contain more streams from pages with no personalized content, such as top lists and playlists created by other users.

5

Conclusion and future work

In this thesis we have investigated the possibility of combining machine learning with data from user logs, in order to gain insights about application usage. This was done through a proof of concept implemented in collaboration with Spotify, a music streaming service, where the task was to identify how users behave in the application when they save music. Through different experiments we were able to show that it is possible to identify limited characteristics of usage, and that it is possible to do so using recurrent neural networks (RNNs). We also show that RNNs with residual connections (RRNs) perform at least as well as RNNs without residual connections.

By modeling the problem as a supervised classification problem, and using state-of-the-art model architectures for sequential learning, we were able to confirm that RRNs can handle a high dimensional feature space of categorical data. We also show that models with increased categorical input complexity have a tendency to perform better than models with lower categorical input complexity in the classification task. The experimental results also indicate that additional numerical features can be added to the categorical feature space in order to increase classification accuracy. However, we suspect that the numerical features require different preprocessing techniques than the ones chosen, and suggest this as possible future work.

By further analyzing the classification results we were able to find indicative feature values and session characteristics that, to a limited extent, describe patterns found by the classifier. The method of analysis included further experiments to determine feature relevance. This was followed by an exploration of the most relevant features, comparing differences and similarities of feature values and session characteristics between the two classes. Note that no attempts were made to locate sequential patterns such as reoccurring sub-sequences of certain elements, although is is a potential extension of our work. The findings show some regularities in sessions where users save music in the application. One such finding is that certain application pages were only represented in sessions classified as positive, which did not always correspond to the representation of pages in the actual positive sessions. Another finding is that the average ms played was higher among sessions classified as negative.

Future work could include further research with different preprocessing techniques for mixed data types. It could be beneficial for model performance if, for example, the numerical features were standardized instead of normalized, i.e. scaled to have

zero mean and unit variance. Furthermore, different methods could be explored to investigate the pattern found by the RRN. One possibility is to gather cell activation statistics [11] in order to investigate if it is possible to locate sequential patterns such as reccurrent sub-sequences.

Despite fairly good classification results, it is not obvious that using such complex models as RNNs was as effective in our setting as initially expected. We are unsure whether the order of events in a sequence is of as great importance as originally anticipated, since no key findings support that assumption, and no attempts at detecting recurrent sub-sequences of events were made. Even though a small scale test with a linear classifier resulted in worse classification accuracy than that of the RNNs, the performance gain is not necessarily worth the time consuming process of trying to understand the underlying model of a complex classifier.

It is highly questionable whether the use of a classifier to discover patterns in artificially labeled data was a suitable method. Given the results, and the lacking analysis of the importance of the sequential order of events, it is clear that the approach used in this thesis was not as suitable as estimated. It is likely that most of the findings presented could have been discovered in a simpler way by statistical analysis and feature selection between the classes of the labeled data. To investigate the usefullness of a classifier in this setting, we could have compared the patterns found by the RNNs to what could be found by statistical analysis. If the classifier had discovered patterns that the plain statistics had missed, it could have been a possible indication that the classifier indeed was necessary, but as it is, we do not have support for such a claim.

Bibliography

[1] M. Kuniavsky, *Observing the user experience: a practitioner's guide to user research*. Morgan kaufmann, 2003.

[2] "The AI disruption wave," https://techcrunch.com/2016/10/13/the-ai-disruption-wave/, accessed: 2017-03-05.

[3] B. D. Davison and H. Hirsh, "Predicting sequences of user actions," in *Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, 1998, pp. 5–12.

[4] J. J. Lee, R. McCartney, and E. Santos Jr, "Learning and predicting user behavior for particular resource use." in *FLAIRS Conference*, 2001, pp. 177–181.

[5] C. Rudin, B. Letham, A. Salleb-Aouissi, E. Kogan, and D. Madigan, "Sequential event prediction with association rules." in *Conference on Learning Theory*, 2011, pp. 615–634.

[6] Y. Wang and F. Tian, "Recurrent residual learning for sequence classification," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

[7] A. Graves, "Supervised sequence labelling," in *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[9] J. Xie and S. Coggeshall, "Prediction of transfers to tertiary care and hospital mortality: A gradient boosting decision tree approach," *Statistical Analysis and Data Mining*, vol. 3, no. 4, pp. 253–258, 2010.

[10] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with lstm neural networks," *arXiv preprint arXiv:1612.02130*, 2016.

[11] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.

[12] P. Munro, *Backpropagation*. Boston, MA: Springer US, 2010, pp. 73–73. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-30164-8_51

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, pp. 214–220, 1988.

[14] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[15] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[16] V. Quintana and E. Davison, "Clipping-off gradient algorithms to compute optimal controls with constrained magnitude," *International Journal of Control*, vol. 20, no. 2, pp. 243–255, 1974.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[18] Q. Liao and T. A. Poggio, "Bridging the gaps between residual learning, recurrent neural networks and visual cortex," *CoRR*, vol. abs/1604.03640, 2016. [Online]. Available: http://arxiv.org/abs/1604.03640

[19] K. M. Ting, "Precision and recall," in *Encyclopedia of machine learning*. Springer, 2011, pp. 781–781.

[20] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.