Learning Purchase Behavior using Site-Centric Clickstream Data and Deep Learning Methods

Robin Bredo
STUDENT NUMBER: 2000623

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

Thesis committee:

Supervisor: dr. G. Cassani
Second Reader: dr. E. Keuleers

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
January 2021

Preface

In this preface I want to express my gratitude towards a number of persons, whom without I would not be able to present this thesis to you as it is right now. I would like to thank dr. Jacobo Tagliabue, for making the Tooso Fashion Clickstream Dataset available to me and my fellow students. Using this dataset gave me valuable insights in working with real data. Besides, I want to thank my supervisor Giovanni Cassani, for giving me the constructive feedback when it was needed. Especially his knowledge in how to process the data and the information you can gain from it, has really helped lifting this thesis to another level. For all the programming that was necessary to conduct this research, I want to thank my fellow students, and in particular Tessa Voorhorst. We really helped each other during this challenging task. The literature, and in particular the articles from Machine Learning Mastery, were really helpful to understand the things needed to be done. Last, but definitely not least, I want to express my gratitude towards my mental support group. My friends, family and partner, who had to listen to all of my frustrations during this time, really helped me in trying to cheer me up when things got difficult.

Hereby I proudly present to you my Master Thesis submitted in partial fulfillment of the requirements for the MSc Data Science & Society degree.

Enjoy reading,

Robin Bredo,
Tilburg, January 2021.

Learning Purchase Behavior using Site-Centric Clickstream Data and Deep Learning Methods

Robin Bredo

It is stated that product information is a valuable indicator of online purchase behavior. In this research it is therefore investigated whether or not product information improves the model performances of neural networks that predict online purchase behavior. Besides, it is tested whether or not this holds for different kinds of neural networks. Two Multi-Layer Perceptrons and two Long Short-Term Memory models were trained on a dataset containing event sequences and product sequences, coming from a clickstream dataset of a major-ecommerce company. Because of several limitations happening during the investigation, the results could not provide valuable and accurate insights on the influence of product information. Future research should implement different methods to aim for a better prediction of online purchase behavior.

1. Introduction

1.1 Context

In this emerging digital world, online shopping has become a daily activity for many people. But as much as an online shopping sessions leads to a purchase being done, very often it also leads to a session in which the customer only looks at products (i.e., "window-shopping") (Tagliabue et al. 2019). According to Bigon et al. (2019), most of the website users have a weak buying intention, which means that a lot of online visits result in window-shopping sessions. Because of this, a key priority to e-commerce companies is to turn window-shoppers into buyers.

To accomplish this, online advertisers should advertise to website users with advertisements that are based on their previous purchases and online activities within particular product categories (Lukose et al. 2010). When users are receiving specific, personal advertisements based on their own preferences, purchase conversion rates will increase. From a marketing perspective, it is therefore important to investigate whether product information improves the performance of models that predict purchase behavior, i.e. predict whether a website user will be a buyer or a window-shopper.

Predicting purchase behavior can be done in many different ways. In this research, purchase behavior is predicted by the use of clickstream data. A clickstream can be seen as a sequence of web pages visited by a user, or a sequence of events done by a user on a website (Jenkins 2019). Using clickstream data to predict purchase behavior is nowadays a popular research subject. Bigon et al. (2019) predicted purchase behavior with clickstream data and despite that they conducted a well-modelled research in which they found valuable results, they also suggested that models should be adapted by adding further metadata, such as metadata concerning the products that the users viewed in their session, i.e. product information.

To the best of our knowledge, no research has been done on the influence of product information on the prediction of purchase behavior. Besides the practical relevance to conduct such a study, it is clear it has a theoretical relevance too. For that reason, this research will focus on adding product information to a model that predicts purchase behavior.

Deep Learning methods are the most common algorithms to use when processing sequential data. Such methods allow for neural networks that are composed of multiple layers, to learn representations of the data with a certain level of abstraction (LeCun, Bengio, and Hinton 2015). By using the backpropagation algorithm, the neural network learns how to change its internal parameters to compute a representation in each layer based on the representation in the previous layer. Each neural network, however, behaves and is composed differently. It is therefore interesting to investigate how different neural networks will perform on similar sequential data. For sequence prediction, the most used neural networks are the Feed-Forward Neural Networks (FFNN) - in particular the Multi-Layer Perceptron, the Convolutional Neural Networks (CNN), and the Recurrent Neural Networks (RNN) - in particular the Long Short-Term Memory (Shickel and Rashidi 2020). As CNNs are more often used for image processing tasks, it is chosen to compare a FFNN with a RNN in this research.

One thing that should be mentioned beforehand is that for this research we will be using a dataset that does not contain specified product information, which means there is, for example, no information concerning the kind of products there are. For each product there is a unique product identifier (a recorded hash), but additional information is lacking. Since there are only product identifiers available, the results will not give insights about the kind of products the users buy or view. However, we may be able to learn something about the product information by creating product embeddings that project similar products to each other into the same representation (Sheil, Rana, and Reilly 2018). Those product embeddings can make the model more accurate, hence perform better. The use of embeddings will be explained in detail in Section 3.2.3. Thus, there will be some limitations regarding the addition of product information to the models, yet the results can be interesting. If we learn that product information provides for a better prediction when they are represented in embeddings, it will be interesting for future research to analyze this further with the appropriate information.

1.2 Research Questions

For this research, this leads us to answering the following research question:

> *To what extent does product information improve the performance of models that predict purchase behavior, and how does this fluctuate between different kinds of neural networks?*

To answer this question, multiple neural networks will be performed, in- and excluding product information, and eventually compared to each other. In particular, we will be comparing the results of four different models: two times a feed-forward neural network (either with or without product information) and two times a recurrent neural network (either with or without product information). In this way, two things will be learned from this research:

- Whether or not product information provides for a better prediction of purchase behavior;

- Whether or not this holds for a particular neural network model, or for different kinds of neural network models.

The research question is divided into multiple sub-questions, in order to conduct the research point by point. The sub-questions are as follows:

RQ 1: When looking at differences between FFNNs and RNNs, is it likely that one network will perform better than the other when processing sequential data?

RQ 2: Does the Feed-Forward Neural Network including product information perform better than the Feed-Forward Neural Network excluding product information?

RQ 3: Does the Recurrent Neural Network including product information perform better than the Recurrent Neural Network excluding product information?

RQ 4: Which of the four models implemented has the best model performance?

2. Related Work

In this section, there will be some more elaboration on the different concepts that were shortly described in the previous section. Particularly, the concepts of clickstream data, purchase behavior, product information and neural networks will be explained in detail.

2.1 Site-centric clickstream data

Clickstream data analysis for e-commerce purposes can, in general, be conducted with two different kinds of clickstream data: user-centric clickstream data, or site-centric clickstream data. User-centric clickstream data consists of a users' online activities on multiple websites, leading to a focus on the entire online experience of a user (Lukose et al. 2010). Features within a user-centric clickstream dataset include: visiting a retail website, visiting a review website, number of different websites visited. Site-centric clickstream data is data that is extracted from one single website. This sort of data can be seen as a subset of user-centric clickstream data. Within site-centric clickstream data, one can find features such as the number of pages viewed, the type of event or the number of clicks within an online activity (Padmanabhan, Zheng, and Kimbrough 2001).

For this research, a site-centric clickstream dataset will be used, which means we are looking at the online activities of users within a particular website, in this case the website of a major European fashion e-commerce company. In Section 3.1 the dataset is described in detail.

According to Lukose et al. (2010), there are two motivations to use site-centric clickstream data when analyzing online purchase behavior: 1) improving web server management by predicting the content the user is likely to request, and 2) to present personalized content based on previous preferences of the user in a particular website. Amazon.com, for example, records the actions of the users and analyzes the browsing history of users to make product recommendations for each individual website user. They recommend items based on the products the users have previously viewed or purchased. Amazon makes these recommendations by analyzing the activities of groups of users who viewed or purchased similar products (Lukose et al. 2010).

2.2 Purchase behavior

By analyzing site-centric clickstream data, we can learn more about the online purchase behavior of customers within a particular website. (Online) purchase behavior is defined here as the "decision processes and acts of online users involved in purchasing or viewing products on a website" (Park and Kim 2003). The prediction of purchase behavior - the primary focus of this research - is defined as a binary classification task, i.e.: the classification of website user sessions as either a buying session or a window-shopping session.

Predicting purchase behavior can be a difficult task to conduct, because of numerous reasons. For example, window-shoppers and buyers appear to be very similar, right up until a purchase occurs (Koehn, Lessmann, and Schaal 2020). Because of the thin line that exists between windows-shoppers and buyers, it can be challenging for a model to separate the two different cases appropriately. Besides that, because website users mostly have a weak-buying intention, the ratio between window-shoppers and buyers is extremely imbalanced, and can be, according to Koehn, Lessmann, and Schaal (2020), 20:1 or higher in favor of the window-shoppers. Whenever a dataset is highly imbalanced, which means that there is a disproportionate ratio of observations in each class, it can become very difficult for a model to provide an accurate prediction because most of the algorithms work under the assumption that classes have an equally number of classes (Brownlee 2019b). This results in models predicting badly, especially for the minority class.

Despite the difficulty, previous research shows that analyzing purchase behavior comes with valuable results. For example, the findings of the research of Benabderrahmane, Mellouli, and Lamolle (2018) support the claim that features that are extracted from clickstream data, convey important information for predicting online purchase behavior. They found that a combination of aggregated statistics and session information result in an accurate and scalable system that predicts online purchase behavior. Bigon et al. (2019) also predicted purchase behavior as a binary classification and found, by modelling a *Long Short-Term Memory Model* on clickstream data, that users have very distinctive clicks right before completing a purchase. If it is known when and why users decide not to purchase a product at that moment, based on their clicking behavior, online e-commerce companies can build upon this information by presenting recommendations and personalized advertisements to try to convince users not to be just a window-shopper, but actually purchase something.

2.3 Product information

According to Lukose et al. (2010), people increasingly use the Internet to search for information about products. Users go online to determine what kind of products are available, in order to fulfill their needs. As to in-store shopping situations, Tellis and Gaeth (1990) found that product information strongly influences purchase behavior, because consumers tend to depend on precise and comprehensible product information in their decision-making process. Kowatsch and Maass (2010) researched whether the claim Tellis and Gaeth (1990) made, also holds for online purchase situations. They found that product information can be valuable with the use of recommendation systems, because these systems explicitly, and implicitly, evoke the desires and expectations of individuals consumers for goods, and make recommendations accordingly based on the product information (Kowatsch and Maass 2010). In this way, recommendation systems

and personalized advertisements can become adaptive and relevant to a customer's individual information needs.

Arora and Warrier (2016) learned vector representations of clusters of products by using an embedding method designed for Natural Language Processing tasks. Such an algorithm tries to learn association between words in a text corpus. This can also be used for clickstream data. Arora and Warrier (2016) used the clickstream sessions as documents and the product attributes as words. The aim of their research was to predict which cluster of products a user is likely to browse in a current session. The results show that learning vectors from product clusters is a valuable way of predicting the context of a website. An embedding method can therefore be seen as a proper method to analyze the product information from clickstream data for this research. This particular method will be used to process the product information and will be explained in detail in Section 3.2.3.

2.4 Neural Networks

Neural networks are computing systems that consist of a collection of connected neurons (also called nodes or units) (Müller, Reinhardt, and Strickland 2012). Each connection can transmit a signal to other neurons. That signal gets processed by the neuron and the neuron signals it again to other neurons connected to it. Neurons are aggregated into layers; these different layers perform different transformations on the signals they are receiving. Signals typically travel from the first layer (i.e., the input layer), to the last layer (i.e., the output layer). Between these two layers a neural network can contain multiple hidden layers in which the signals are traversed before they end up in the output layer.

The training of a neural network goes as follows: examples, that contain either a known input or a known result, are processed by forming associations between the inputs and the results. The difference between the processed output of the network (often a prediction) and the target output (what needs to be predicted), which is known as the error, is determined. The network adjusts the associations by the use of a learning rule in order to reduce that error value. If the adjustments are successful, the neural network is able to produce output which is increasingly similar to the output that needs to be produced (the target output) (Müller, Reinhardt, and Strickland 2012).

It is shown that neural networks have been achieving great success in Natural Language Processing tasks and in particular sequence classification (Zhao et al. 2017). This is because neural networks are able to transform the original input space into a feature space. In such a feature space it is easier to separate different classes of the input data, and therefore provide for a better classification performance (Müller, Reinhardt, and Strickland 2012). For example, Bigon et al. (2019) predict purchase behavior by modelling clickstream data onto a Long Short-Term Memory model; a recurrent neural network especially built for sequence classification. Their LSTM outperformed a Naive Bayer Classifier and a Markov Chain Model. Wu et al. (2015) built a bidirectional LSTM; a recurrent neural network that has connections in both forward and backward direction in time. Their model outperformed a Gradient Boosting Regression, and a simple, deep neural network.

For this research it is chosen to model Feed-Forward Neural Networks and Recurrent Neural Networks to the sequential data, to compare whether a particular model would perform better than the other. In the sections below we get into detail about FFNNs and RNNs and elaborate on their differences in implementation and performances.

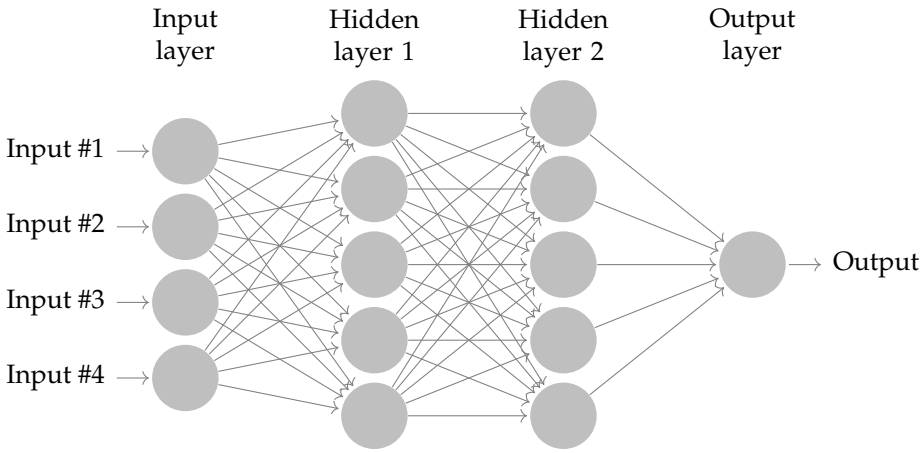

Figure 1
Visualization of a Multilayer Perceptron with two hidden layers
Adapted from: StackExchange (2017)

2.4.1 Feed-Forward Neural Networks (FFNNs). Feed-Forward Neural Networks are artificial neural networks in which the connections do not form a cycle, but are feed-forward: the information moves only in one direction - forward - from the input layer to the hidden layer (if any), and eventually to the output layer (Schmidhüber 2015). Every unit in a (hidden) layer that is defined in the network is fully connected with all the units in the next layer.

Despite the fact that Recurrent Neural Network are often preferred over a FFNN when processing sequential data, it is shown that FFNNs are also suitable for this task. For example, Benabderrahmane, Mellouli, and Lamolle (2018) predicted purchase intention by the use of aggregated pageview data that was kept track during the user's visit, together with session and user information. They found that an Multi-Layer Perceptron (MLP) achieved the best results and outperformed Decision Trees and Support Vector Machines.

A MLP is a feed-forward artificial neural network made up of multiple (hidden) layers of nodes in a direct graph. MLPs are often used to model nonlinear problems because they can work with nonlinear activation functions in the hidden layers (Sakar et al. 2019). MLPs are widely used for classification prediction problems where the inputs are assigned to a class or a label. They are capable of reducing sequences into a long row of data and then perform, for example, a classification task (Brownlee 2018b). This makes MLPs also suitable for the data used in this research. In Figure 1, a MLP with two hidden layer is visualized.

2.4.2 Recurrent Neural Networks (RNNs). Recurrent Neural Networks can be seen as a family of neural networks that are designed especially to process sequential data (Goodfellow, Bengio, and Courville 2016). While a FFNN is limited to having only feed-forward connections, RNNs have feed-forward connections, self connections, and backward connections (Benabderrahmane, Mellouli, and Lamolle 2018). They can use their internal states to process variable length sequences as input. This makes them suitable for sequential information; they take what they have received from previous layers as input.

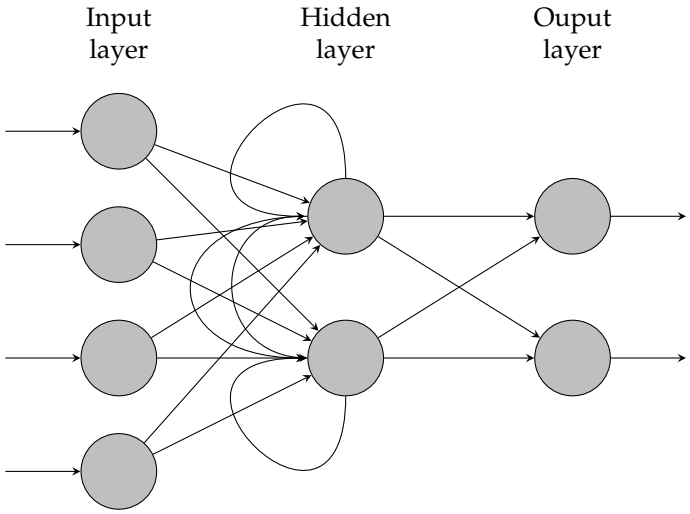

Figure 2
Visualization of a Recurrent Neural Network
Adapted from: StackExchange (2019)

Because feed-forward neural networks like the Multi-Layer Perceptron are restricted to only forward connections, such networks are not capable of capturing temporal dependencies in sequential data. As a solution to this problem, recurrent neural networks were introduced. The connections in these kind of neural networks facilitate that ordering information is included into the network (Koehn, Lessmann, and Schaal 2020). In short, the underlying idea of a RNN is that an input sequence is connected to a recurrent layer, which is connected to itself and also to an output layer. In Figure 2, a RNN is visualized.

Recurrent neural networks are in theory able to capture contextual dependencies over an infinite time-lapse, but Bengio, Simard, and Frasconi (1994) argue that in practice this comes with many difficulties. When the network tries to learn long-term dependencies, the values of the gradient for many time steps back can become marginally small (i.e., *vanishing gradient problem*), or they explode (i.e., *exploding gradient problem*) (Toth et al. 2017). When gradients vanish, learning the data becomes difficult because the correct tracking of the gradient is difficult to discern, while learning the data becomes unstable when gradients explode. Both are undesirable outcomes.

To solve this, Hochreiter and Schmidhuber (1997) introduced the *Long Short-Term Memory Model* (LSTM). In a LSTM, the nodes are defined by memory cells, instead of ordinary recurrent units. These memory cells consist of four gates: an input gate, an input unit with a self-recurrent gate, a forget gate, and an output gate. The idea behind the LSTM model is that it is able to learn when to forget older states, which will help to alleviate the effect of the vanishing gradient problem (Jenkins 2019). The function of the memory cells is that some connections can retrieve information from them, while other connections will cause them to forget.

2.5 Differences between FFNNs and RNNs

It is argued that modelling RNNs - and in particular the LSTM - onto clickstream data, provides for a better prediction of purchase behavior, than modelling FFNNs. Montgomery et al. (2004) claim that in their model, the memory component was crucial in predicting a clickstream path accurately. As clickstream paths may reflect the goals of website users, using a memory component in the model will gain better results. As stated in Goodfellow, Bengio, and Courville (2016), the advantage of the RNN (or the LSTM for that matter) is the capability of remembering previous inputs, in order to improve the performance of predicting time-series data, or sequential data. This is because the model loops through the layers in a sequential way and has therefore a "real sense of time" even before the model is fully trained. Brownlee (2018b) also argues when dealing with data that is represented as sequences, RNNs in general and LSTMs in particular have received the most success.

The memory cells that are built into the RNNs are the biggest advantage of the neural network, opposed to the FFNNs. Especially when sequential data is being processed, the memory cells are a crucial component (Montgomery et al. 2004). This is why it is expected that the RNNs will outperform the FFNNs in this research.

3. Experimental Setup

In this study, the prediction of purchase behavior is designed as a binary classification problem classifying a given user session being either a buying session (label 0) or a window-shopping session (label 1). This task is conducted four times, each time with a different neural network or a different input. In Table 1, the formal definition of each model is described.

In the following sections the dataset that was used is described, along with the preprocessing steps that were took, a small Exploratory Data Analysis, and the models that were implemented to predict purchase behavior. All the analyses are done in the programming language Python and the code was written in Notebooks of Google Colaboratory. The packages used for the analyses and data transformations are listed in Appendix A, together with descriptions and their sources.

3.1 Data

The dataset that was used to predict purchase behavior is the "Tooso Fashion Clickstream Dataset" (Bigon et al. 2019). This is a dataset containing real website user sessions coming from a major European e-commerce fashion website. The website visits of the users took place between the 29th of June and the 18th of July in the year 2018. Any click performed by users browsing on the website is collected in the dataset with an anonymized user ID, a timestamp, and product metadata. Each click can be seen as an event of a certain kind of type (i.e.: view, click, detail, add-to-cart, remove-from-cart, or buy). An example of a particular row in the dataset is represented in Table 2.

3.2 Preprocessing

In this section it is described how and why some preprocessing steps were taking. As a first step, the data was cleaned. Thereafter, the representations of the user sessions and the product sequences were transformed.

Table 1
Formal problem definition of each model

Model	Formal problem definition
1. MLP excl. product information	Given a session X (X_1 ... X_N), classify X as BUY (y = 1) or NO-BUY (y = 0)
2. MLP incl. product information	Given a session X (X_1 ... X_N), and product sequence Z (Z_1 ... Z_N), classify X+Z as BUY (y = 1) or NO-BUY (y = 0)
3. LSTM excl. product information	Given a session X (X_1 ... X_N), classify X as BUY (y = 1) or NO-BUY (y = 0)
4. LSTM incl. product information	Given a session X (X_1 ... X_N), and product sequence Z (Z_1 ... Z_N), classify X+Z as BUY (y = 1) or NO-BUY (y = 0)

Table 2
Example of a row in the raw data

Variable Name	Random Example
session_id_hash	00000005c19449b57d8d05dfc8b5474be0654032
target_user_id_hash	NaN
client_user_id_hash	a04b785089851dfd30c6a849f96e2b1827396303
event_type	pageview
product_action	detail
product_skus_hash	[E98D8BCA8E4B56DD]
server_timestamp_epoch_ms	1530998447263
server_date	2018-07-07
hashed_url	BB59BC179D3D58F4

3.2.1 Data Cleaning. The consecutive events within the same session were transformed into an event sequence. This led to a dataset in which each row represented a user session with the consecutive events as a sequence, opposed to the raw dataset in which each row was represented by a single event. Sessions with a length of less than five events and sessions with a length of more than 200 events were ommitted from the dataset. Because these sessions are extremely short and extremely long, they are seen as 'not human'. In this way, all sessions contained enough meaningful information.

Besides that, the 'purchase' events were removed from the sessions, since that type of event is the target label that needs to be predicted by the model. Each purchase session was labeled as 1 and each window-shopping session was labeled as 0. This data

Table 3
Number of data examples before and after data cleaning

Feature	Before data cleaning	After data cleaning
Total events	5.433.606	4.277.528
Total sessions	443.656	203.958
Total unique products	38.345	36.256

cleaning led to a different number of events, sessions and products. This is shown in Table 3 below.

3.2.2 Representation of the user sessions. Because machine learning algorithms cannot work with categorical data directly - they only allow numeric data to be fed into the model (Brownlee 2019a) - the representations of the user sessions and the products needed to be transformed from categorical representations to numerical representations.

The first thing that needed to be done was to integer encode the events. This was done in the following way:

1. The event "view" is encoded as 1
2. The event "detail" is encoded as 2
3. The event "add" is encoded as 3
4. The event "remove" is encoded as 4
5. The event "click" is encoded as 5
6. The event "purchase" is encoded as 6 [1]

At this point, the events are integer encoded but still seen as some sort of categories: they are now ordinal data, in which the event coded as 4 is assumed to weigh double the event coded as 2, and four times as much as the event coded as 1. In order for the models to work, the sessions needed to be nominal, where there is no order between the event types. That is why the events are one-hot-encoded. One-hot-encoding allows for a more expressive and numerical representation of the categorical data, represented as binary vectors. (Brownlee 2017a). Each value that stands for an event in the sequence (1-5) is a binary vector with six classes, all zero values except for the value that represents the event, which is marked with a 1. A few examples of how the events were eventually fed into the models - represented as sessions - are displayed in Table 4. Eventually, the one-hot-encoded sessions were zero-padded to the longest event sequence, which was 179 events long. This padding method is applied because neural networks require sequences of the same length (Goodfellow, Bengio, and Courville 2016).

1 Note that the purchase events are removed from the sessions, as it is the target label.

Table 4
Encoded event sequences

Categorical	Integer-encoded	One-Hot-Encoded
[view, view, detail, view, detail, view]	[1, 1, 2, 1, 2, 1]	[[0. 1. 0. 0. 0. 0.] [0. 1. 0. 0. 0. 0.] [0. 0. 1. 0. 0. 0.] [0. 1. 0. 0. 0. 0.] [0. 0. 1. 0. 0. 0.] [0. 1. 0. 0. 0. 0.]]
[view, view, view, view]	[1, 1, 1, 1]	[[0. 1. 0. 0. 0. 0.] [0. 1. 0. 0. 0. 0.] [0. 1. 0. 0. 0. 0.] [0. 1. 0. 0. 0. 0.]]
[view, detail, add, remove, view]	[1, 2, 3, 4, 1]	[[0. 1. 0. 0. 0. 0.] [0. 0. 1. 0. 0. 0.] [0. 0. 0. 1. 0. 0.] [0. 0. 0. 0. 1. 0.] [0. 1. 0. 0. 0. 0.]]

3.2.3 Representation of the product sequences. By transforming the events into sessions, the products also appeared in the sequence, ordered in the way they were clicked on during a website visit. It could occur that a product was not recorded. When this happened, an empty list was created. So, a product sequence could look like: [[], [], ['24846F9ABAC33A0A'], []], where ['24846F9ABAC33A0A'] represents a product ID. Because these product sequences are seen as categorical data, they were also one-hot-encoded. Similar as to the user sessions, the one-hot-encoded product sequences were zero-padded to the length of 179.

However, one-hot-encoding can create very sparse vectors. This means that there are many zero values in the vector. This was not a problem regarding the user sessions, because there were no more than 6 values in the vector. For the product sequences however, this could be problematic because those vector were as long as there were unique products. This led to very sparse product sequences, which would lead to the processing of a lot of memory when running the analyses. Neural networks work better when vectors are dense (i.e., when most of the values in the vector are non-zero). Therefore, when building the models, an Embedding layer was added to the model. In this way, the models would require less running time and memory.

The Embedding layer is provided by the Python package *Keras*, which is a package especially for building deep neural networks (Brownlee 2017b). The Embedding layer turns positive integers into dense vectors of fixed size (Chollet et al. 2015). The layer will learn embeddings for all the products in the product sequences. Embeddings are vector space models. These models work by embedding products to nearby point where other, similar, products are mapped (Sheil, Rana, and Reilly 2018). In this way, relationships between products are taking into account. In the embeddings, the products are represented as dense vectors where each vector represents the projection of the product into a continuous vector space. Embeddings are also a way to let your model work faster

and more accurate. For the models to be able to work, including the Embedding layer was found a necessary task in this research.

3.3 Exploratory Data Analysis

To get a good picture of the data, a small EDA was conducted. The results of this analysis are described in this section, alongside with supporting visualizations. The analysis is based on both the raw and cleaned data.

In Figure 3, the distribution of the product actions is shown (i.e., how many times a certain action on a product occurred). Every time a user views a product, the user often also viewed the details of the product (product action 'detail'). Only a small amount of the recorded products is bought, even though a bigger amount was added to the cart. The absolute amounts are shown in Table 5.

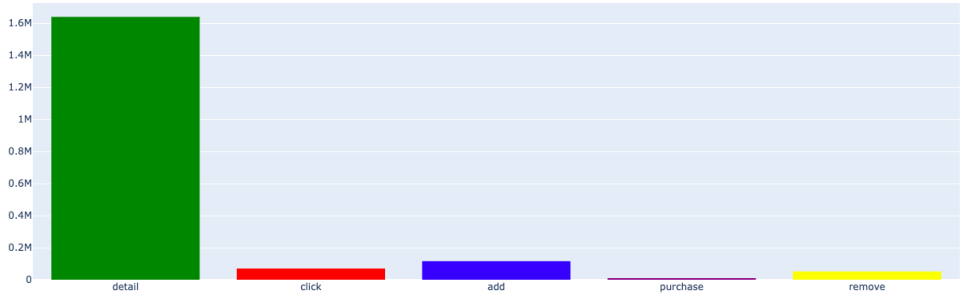

Figure 3
Distribution of the product actions

Table 5
Distribution of the product actions

Product action	Count
Detail	1.639.238
Click	69.828
Add to cart	115.555
Purchase	9.920
Remove from cart	51.512

In Table 6 it is shown which products are bought, added and removed the most and how many times this occurred. Independently of whether or not the addition of product information provides for a better classification in predicting purchase behavior, the information in the table can already be of value.

There are a few products that ended up in multiple columns. For example, the two most viewed products are also the two products that were added to the cart most often. However, these products did not occur in the top five most bought products. The product that is bought the most, is also one of the most removed products. In this overview cart abandonment also becomes clear: there are way more products added to the cart once, than that they are bought or removed, which indicates that the products are left in the shopping cart.

Table 6
Products that are bought, added to cart, and removed from cart the most. In the columns # Viewed, # Bought, # Added, and # Removed it is stated how many times the product action occurred. Products that are presented in italics are products that occur multiple times in the table.

Product Hash	# Viewed	# Bought	# Added	# Removed
[590A2FAB43CBE2CF]	9152			
[77B8A76CB507265A]	7677			
[4F72DD11C6C70C46]	5231			
[372350A2C8561B20]	4749			
[D0343A64953DD2AA]	4573			
[30B54C2800CF7EB1]		14		
[53D54C573A4508E3]		12		
[BD6B1CC5883C333B]		11		
[D11181799588D257]		10		
[3E0594FCA1A27E07]		9		
[590A2FAB43CBE2CF]			962	
[77B8A76CB507265A]			452	
[0D97729AE1A005E8]			377	
[9032FBD5CD9D24D0]			300	
[7B91AD1C2B4FE193]			281	
[F91053126E62BA58]				79
[F7C2306A30A82910]				69
[30B54C2800CF7EB1]				60
[8D3333CCD0F74D81]				50
[662CCF904D446BDF]				47

3.4 Class imbalance

By exploring the data it became clear that the two classes (buying vs. window-shopping) are highly imbalanced. This is something that has been noticed in previous research too. For example, Bigon et al. (2019) stated that website visitors often have a weak buying intention which indicates that most website visits result in window-shopping sessions. Koehn, Lessmann, and Schaal (2020) showed that in their research the distribution between window-shoppers and buyers was highly imbalanced too, with a disproportionate ratio of 20:1 in favor of the window-shoppers. In this dataset, only 4.2 percent of the sessions led to a purchase.

Class imbalance can cause problems to the performance of classification models. For example, the evaluation metric that is often used for classification problems is accuracy. However, when using accuracy for an imbalanced classification, the metric tends to favor the majority class, resulting in a high accuracy score while the model actually performed badly (Weng and Poon 2008). The AUC-score is a more appropriate choice of metric when dealing with imbalanced classes. This metric does not bias on the size or the (im)balance of the different classes.

Another way to deal with class imbalance is to resample the classes so that the distribution between the classes becomes equal. This is done prior to the implementation of the models. There are many different ways to resample your data. For this research,

it is chosen to use the Synthethic Minority Oversampling Technique (SMOTE) in order to get a proportionate ratio of the two classes. This method uses a nearest-neighbors algorithm to generate new, synthetic samples for the training data (Boyle 2019). It is an oversampling technique which means that the minority class will receive more, new samples in order to get an equal number of samples as there are in the majority class. It is important to generate these new samples before the data is split into training and testing data, in order for the models to generalizes well to unseen data (Chawla et al. 2002). The sampling strategy was set to 'auto', meaning that the algorithm will oversample the minority class and random state was set to 27. This holds for all four models.

3.5 Models

In this section, the neural networks that were built for this research are described in detail. All networks are made with the use of the Python *Keras* package (Chollet et al. 2015). In every model, the data was split into 80% training set and 20% testing set, using the train_test_split() function provided by the Python *sci-kit learn* package (Pedregosa et al. 2011). This is a common ratio, known as the 80-20 ratio or Pareto principle (Dunford, Su, and Tamang 2014).

3.5.1 General Parameters. To determine the output of the neural networks, four general parameters were implemented in each of the four models: an activation function, loss function, optimizer and early stopping method.

Activation function. An activation function makes the network more dynamic and able to extract complex information from the input data. The most widely used activation function is the ReLU function, which stands for Rectified Linear Unit, and performs, according to Sharma (2020), better than other functions in most of the cases. The ReLU activation function is non-linear and resolves the vanishing gradient problem causing the network learn faster and better. One disadvantage of the ReLU activation function is that it cannot be activated in the final layer (the output layer). Another activation function that works well with classification tasks is the Signoid activation function (Sharma 2020). However, if the Sigmoid activation function is used in hidden layers, the vanishing gradient problem cannot be solved. Therefore, it is common to activate a ReLU function in hidden layers and use a Sigmoid activation function in the output layer.

Loss function. The loss function that was specified for each of the four models, is the Binary Cross-Entropy loss function. This is done because of the binary classification task that holds for this research. The Binary Cross-Entropy loss function is a combination of the Sigmoid activation and a Cross-Entropy loss. Its loss is independent for each vector component, meaning that the loss that is computed for every output vector is not affected by other component values (Janocha and Czarnecki 2017). That is why it is a common loss function for classification tasks: the interpretation of an element belonging to a certain class should not affect the decision for another class.

Optimizer. As optimizer, the 'Adam' optimizer was specified in all models. Optimizers are used to change the attributes of the neural network, such as the weights and the learning rate, in order to reduce the losses. Results coming from the neural network will come in a faster way. The 'Adam' optimizer is an algorithm for the optimization of first- and second-order movements in the network. It is a stochastic gradient descent

method based on the adaptive estimation of the movements. According to Kingma and Ba (2014), the method is computationally effective, requires little memory, is invariant to rescaling gradients diagonally, and is well suited for large data/parameter issues.

Early Stopping. The Early Stopping method was also implemented in each of the four models. This is done to prevent the model from overfitting or underfitting. It is often a difficult task to decide how many epochs your model needs to run. This is where the Early Stopping algorithm comes in handy. Early Stopping stops the training once the model performance stops improving (Brownlee 2018a). Early Stopping was defined using the *Callbacks* function provided by Keras. For each model, the "monitor" parameter was set to the loss function (Binary Cross-Entropy). This means that the algorithm monitors the performance of the loss; when the loss is not decreasing anymore (stops improving), the algorithms stops training. The "patience" parameter was set to three, to make sure that the model will not be stopped at the first time the performance is not improving anymore. A model can get worse before getting better, so ending the training at the first glimpse of no improvement may not always be a good idea (Brownlee 2018a).

3.5.2 Model 1: Multi-Layer Perceptron excluding product information. The first model that was built, was a MLP with the event sequences (i.e., the one-hot-encoded user sessions) as input. As in all four models, the task was to classify the sequences as buy (label 1) or window-shopping (label 0). In this model, three Dense layers were implemented, which means the first Dense layer is the input layer with 32 output neurons, the second layer is a hidden layer with 16 output neurons and the third layer is the output layer with the number of classes as output neurons. The information coming from the first layer is funneled into a dense format in each next layer. This makes the model capable of learning the important patterns. A Dense layer is as fully connected layer, meaning that all the neurons in the layer are connected to those in the next layer (Brownlee 2016). It is the most common layer used for MLPs. In every Dense layer, the ReLU activation function was implemented and the model was closed with an Activation layer in which the Sigmoid function was defined. The model was compiled with a Binary Cross-entropy loss function and the 'Adam' optimizer.

When fitting the model onto the training data, ten epochs were initialized, together with batches of size 32 and a validation set of 10%. An epoch is seen as one pass through all the rows in the training dataset, and a batch is a sample (or samples) considered by the model within an epoch, before the weights in the network are updated. Based on the chosen batch size, one epoch is thus composed of one or more batches. In Table 7 the construction of the different layers in Model 1 is clearly displayed.

3.5.3 Model 2: Multi-Layer Perceptron including product information. To check whether product information would provide for a better classification, another MLP was built, now including the product sequences (i.e., the one-hot-encoded product hashes). In order to do this, two separate models were built in first instance, that were later concatenated to each other. The first model, model x, is a MLP with the event sequences as input. The model consisted of an Embedding layer, an Flatten layer and three Dense layers. Despite that the Embedding layer should not be necessary in this model, because the one-hot-encoded event sequences are relatively small with respect to the dimensions, it was included in order to combine the two models and make the model run faster. The input dimension of the Embedding layer was set to six (as there are six dimensions in the encoded event sequences), the output dimension was set to 32, and the input length was set to the maximum sequence length, which was 179.

Table 7
Model Summary of Model 1

Multilayer Perceptron excluding product information		
Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
dense (Dense)	(None, 32)	34400
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17
activation (Activation)	(None, 1)	0
Total params: 34,495		
Trainable params: 34,495		
Non-trainable params: 0		

The output of the Embedding layer is a two-dimensional vector. In order to connect a Dense layer directly to the Embedding layer, a Flatten layer is required to flatten the two-dimensional vector to a one-dimensional vector. Similar to Model 1, in every Dense layer the ReLU activation was specified. It was chosen to use three Dense layers - which is next to the Embedding and Flatten layer a high number of Dense layers - in order to create an as much as possible similar model as Model 1.

The second model of this concatenated MLP, model *y*, has the same construction as model *x*: it consisted of an Embedding layer (now with an input dimension as big as there are dimensions in the encoded product sequences, an output dimension of 32 and input length of 179), an Flatten layer and three Dense layer with an ReLU activation function. To concatenate the two models, the *concatenate()* function provided by Keras was used. The output layer was defined as a Dense layer with a Sigmoid activation layer. The final model was compiled with a Binary Cross-Entropy loss function and the 'Adam' optimizer.

When fitting the model to the training data, the event sequences and product sequences were the combined input. Similar to Model 1, the number of epochs was set to ten, the batch size was 32 and there was a validation set of 10%. In Table 8, the construction of the final model is made visible. Because two models were concatenated with each other, and both models were constructed the same, this led to a model consisted of many hidden layers, as seen in the table.

Table 8
Model Summary of Model 2

Multilayer Perceptron including product information

Model: "functional"

Layer (type)	**Output Shape**	**Param #**
input_1 (Input layer)	[(None, 179, 6)]	0
input_2 (Input layer)	[(None, 179, 2618)]	0
embedding (Embedding layer)	(None, 179, 6, 32)	192
embedding_1 (Embedding layer)	(None, 179, 2618, 32)	83744
flatten (Flatten)	(None, 34368)	0
flatten_1 (Flatten)	(None, 14995904)	0
dense (Dense)	(None, 20)	687380
dense_3 (Dense)	(None, 20)	299918100
dense_1 (Dense)	(None, 10)	210
dense_4 (Dense)	(None, 10)	210
dense_2 (Dense)	(None, 1)	11
dense_5 (Dense)	(None, 1)	11
concatenate (Concatenate)	(None, 2)	0
dense_6 (Dense)	(None, 1)	3

Total params: 300.689.861
Trainable params: 300.689.861
Non-trainable params: 0

3.5.4 Model 3: Long Short-Term Memory excluding product information. The third model we defined in this research was a Long Short-Term Memory model. As described earlier, the LSTM is a Recurrent Neural Network that was especially designed to resolve the vanishing and exploding gradients that occur when training long-term dependencies on a simple RNN. In this LSTM, we specified the event sequences as input. The LSTM consisted of a LSTM layer, a Dropout layer, a Dense layer and an Activation layer. The number of hidden nodes in the LSTM layer was set by following the formula (Eckhardt 2018):

$$N_h = \frac{2}{3} * (N_i + N_0)$$

where:

N_h is hidden nodes
N_i is the sequence length
N_0 is the number of events

This led to a number of 716 hidden nodes. A Dropout layer was used to prevent the model from overfitting. According to, Brownlee (2017b), a dropout rate of 20% is usually an accurate specification. Therefore, in our model, the dropout rate in the Dropout layer was set to 0.2. In the Dense layer, the activation function ReLU was specified, and in the Activation layer, the Sigmoid function was specified, similar to the other models. When compiling the model, the loss function was set to Binary Cross-Entropy and the

Table 9
Model Summary of Model 3

Long Short-Term Memory excluding product information		
Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
lstm (LSTM)	(None, 716)	2070672
dropout (Dropout)	(None, 716)	0
dense (Dense)	(None, 1)	717
activation (Activation)	(None, 1)	0
Total params: 2.071.389		
Trainable params: 2.071.389		
Non-trainable params: 0		

optimizer to 'Adam'. When fitting the model, ten epochs were fitted, alongside with batches of size 32 and a validation set of 10%. The construction of Model 3 is displayed in Table 9.

3.5.5 Model 4: Long Short-Term Memory including product information. Similar to Model 2, in Model 4 we also concatenated two models with each other, to included the product information. In the fourth model, two LSTMs were combined in order to check whether product information improves the performance of a model predicting purchase behavior. In the first model, model x, we built a LSTM with the event sequences as input.

This model consisted of an Embedding layer, a LSTM layer, a Dropout layer and a final Dense layer. The Embedding layer was defined with the same parameters as the Embedding layer in Model 2(x): input dimensions of six, output dimensions of 32 and input length of 179. The LSTM layer was specified with the same number of hidden nodes as in Model 3: 716 hidden nodes. The dropout rate in the Dropout layer was set to 20% and the Dense layer had a ReLU activation function.

The second model needed to create Model 4, model y, was also a LSTM model with an Embedding layer, LSTM layer, Dropout layer and Dense layer. In this model, the product sequences were defined as input. This model looked similar to model 4(x), except for the input dimensions in the Embedding layer, which was set to 2618 in model 4(y). The rest of the parameters needed for this model, were the same as in Model 4(x). With the *concatenate()* function provided by Keras, the two models were combined into one model, Model 4. This model had an Dense output layer, with the Sigmoid activation function.

Similar to all other models, Model 4 was compiled with a Binary Cross-Entropy loss function, the 'Adam' optimizer and the AUC-score as metric. Ten epochs, batches of sizes 32 and a validation set of 10% were then fitted on the model, with the event sequences and the product sequences as a combined input. In Table 10 the construction of Model 4 is made visible.

Table 10
Model Summary of Model 4

Long Short-Term Memory including product information

Model: "functional"

Layer (type)	**Output Shape**	**Param #**
input_1 (Input Layer)	[(None, 179, 6)]	0
input_2 (Input Layer)	[(None, 179, 2618)]	0
lstm_1 (LSTM)	(None, 100)	42800
lstm_2 (LSTM)	(None, 100)	1087600
dropout_1 (Dropout)	(None, 100)	0
dropout_2 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 1)	101
dense_2 (Dense)	(None, 1)	101
concatenate (Concatenate)	(None, 2)	0
dense_3 (Dense)	(None, 1)	3

Total params: 1.130.605
Trainable params: 1.130.605
Non-trainable params: 0

3.6 Bottlenecks

Before presenting the results of the models, one thing should be mentioned. Unfortunately, we were not able to process the whole dataset. When running the models that included the product information, the sessions crashed because the RAM-memory was not big enough. Despite that there was an Embedding layer included, which transformed the sparse vectors into dense vectors and would led to a faster running-time, the running-time kept crashing when including more data. This is a problem we could not foresee. We tried many things, included transforming the data otherwise, but unfortunately it did not matter. Therefore, in every model, a subset of 15.000 rows was used. This led to a total of 542 sessions (433 in the training set, 109 in the testing set), in which 2617 different products were viewed.

3.7 Evaluation Criteria

To evaluate the performances of the four models, the AUC-metric and the F1-score are used. These are appropriate measurements when the data is imbalanced. Although the data is balanced prior to the modelling of the neural networks, these metrics are still consulted instead of the accuracy score. This is because these scores do not have any bias towards a certain class, whereas accuracy this still might has despite the balanced classes (Weng and Poon 2008). However, because the testing dataset is still highly imbalanced, the F1-score and the AUC-score will be evaluated. The F1-score takes a value between 0 and 1 and is calculated by this formula (Wardhani et al. 2019):

$$F1score = \frac{2 * Precision * Recall}{Precision + Recall}$$

where:

$Precision$ is the proportion of correct positive predictions
$Recall$ is the proportion of actual positives that was predicted correctly

The AUC-score tells how much the model is capable of distinguishing the two different classes. The AUC-score represents the probability that a random positive example is positioned to the right of a random negative example (Weng and Poon 2008). The score is calculated as follows:

$$AUCscore = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} 1_{p_i} >_{p_j}$$

where:

i runs over all m input data with true label 1
j runs over all n input data with true label 0
p_i and p_j denote the probability score assigned to input i and j, respectively.
1 is the indicator function: it outputs 1 if the condition ($p_i > p_j$) is satisfied

4. Results

In this section, the results of Model 1 and 2 will discussed first, followed by the results of Model 3 and 4.

4.1 Model 1 and 2

4.1.1 Results Model 1. The Early Stopping method implemented in the model made the model stop running after the fifth epoch (out of ten epochs), at a loss of 0.4650. The loss is always a value between 0 and 1 where a value of 0 indicates a perfect score, meaning that there are no differences between the predicted value and the actual value. In this model, the AUC-score was 0.636. An AUC-score of 0 would indicate that the predictions of the model are 100% wrong (false positives), where a score of 1 would indicate that the predictions are 100% correct (true positives). When the AUC-score is approximatey 0.5, the model is not capable of distinguishing the positive class instances from the negative class instances. The overall F1-score of Model 1 is 0.853, showing that the model was relatively well at predicting the purchase label. The other scores are displayed in Table 11.

4.1.2 Results Model 2. This model was ran over eight epochs before the Early Stopping stopped the model. After the last epoch, the result of the model was a loss of 0.1980 and an AUC-score of 0.636. In Table 11 and Table 12 it becomes clear that the performance scores of Model 2 are exactly the same as the scores of Model 1. Similar to Model 1, in the tables it becomes clear that the testing set was highly imbalanced, with only seven samples in the minority class, and 102 samples in the majority class. This makes it difficult for the Models to perform well.

Table 11
Model Performance of Model 1 (MLP excluding product information)

	precision	recall	f1-score	support
0	0.96	0.84	0.90	102
1	0.16	0.43	0.23	7
				109
Accuracy of model	0.817			
Precision of model	0.904			
Recall of model	0.817			
F1-Score of model	0.853			
AUC-ROC of model	0.636			

Table 12
Model Performance of Model 2 (MLP including product information)

	precision	recall	f1-score	support
0	0.96	0.84	0.90	102
1	0.16	0.43	0.23	7
				109
Accuracy of model	0.936			
Precision of model	0.876			
Recall of model	0.936			
F1-Score of model	0.905			
AUC-ROC of model	0.636			

4.1.3 Performance. Regardless of the similar performance scores of Model 1 and Model 2 there is a difference in the losses. Model 1 had a loss of 0.4650 which indicates there was a slightly difference between the predicted values and the actual values. Model 2 had a loss of 0.1950 meaning that the predicted values are almost similar to the actual values. By looking at the overall F1-scores of the models, we state that the models are performing well. However, when we look at the F1-scores for each class, the scores indicate something different. For the majority class, the window-shopping sessions, the F1-score for both models is 0.90, meaning that the model was able to predict the samples properly, but the F1-score for the minority class is 0.23, meaning that the model was not able to predict the buying-sessions well.

4.2 Model 3 and 4

4.2.1 Results Model 3. When training this model, The Early Stopping terminated the model from running after the fourth epoch. The model was stopped at a loss of 0.6931 and an AUC-score of 0.636. This loss is higher than the losses of Model 1 and Model 2, indicating that the MLPs did a better job in predicted values similar to the actual values. In Table 13 the different scores of the model performances are shown. The results are the same as those of Model 1 and Model 2.

Table 13
Model Performance of Model 3 (LSTM excluding product information)

	precision	recall	f1-score	support
0	0.96	0.84	0.90	102
1	0.00	0.43	0.23	7
				109
Accuracy of model	0.817			
Precision of model	0.904			
Recall of model	0.817			
F1-Score of model	0.853			
AUC-ROC of model	0.636			

Table 14
Model Performance of Model 4 (LSTM including product information)

	precision	recall	f1-score	support
0	0.96	0.84	0.90	102
1	0.00	0.43	0.23	7
				109
Accuracy of model	0.817			
Precision of model	0.904			
Recall of model	0.817			
F1-Score of model	0.853			
AUC-ROC of model	0.636			

4.2.2 Results Model 4. The last model that was trained, was the LSTM including product information. The training ended after ten epochs, which was the number of epochs that was initialized in the model. After ten epochs, the loss ended up being 0.6461 and the AUC-score was 0.636. See Table 14 for the other performance scores of Model 4.

4.2.3 Performance. Similar as to the performances of Model 1 and 2, there is no clear distinction between the performances of Model 3 and 4. There is only a difference in the loss, with a loss of 0.6931 for Model 3 and a loss of 0.6461 for Model 4. However, this difference is minimal, causing Model 4 to perform similar to Model 3.

5. Discussion

5.1 Research Questions

The goal of this thesis was to investigate whether product information would provide for a better classification of user sessions. Therefore, the following Research Question was determined:

> *To what extent does product information improve the performance of models that predict purchase behavior, and how does this fluctuate between different kinds of neural networks?*

Based on four predetermined sub-questions this question can be answered. In this section, the sub-questions will be discussed first, after which the main question is answered. The first sub-question is as follows:

> *RQ 1: When looking at differences between FFNNs and RNNs, is it likely that one network will perform better than the other when processing sequential data?*

As described in Section 2.4, the literature states that both a Feed-Forward Neural Network and a Recurrent Neural Network are capable of processing sequential data. However, as argued by previous research, RNNs often perform better when classifying sequences. There are a few reasons for this to mention.

When processing sequential data, a FFNN reduces the sequences to a long row of data and then tries to classify the sequences (Brownlee 2018b). This can lead to valuable results, however, when the sequences are reduced to one row, changes are that some information is omitted. Besides that, because FFNNs are restricted to only forwarding inputs - in contrast to RNNs that have recurrent layers - FFNNs are not capable of processing temporal dependencies in sequential data. That is the exact reason why RNNS were designed; they are especially designed to capture the contextual time-dependencies in sequences (Goodfellow, Bengio, and Courville 2016). The recurrent layers that are incorporated in the RNNs are capable of ordering the information that is included into the network (Koehn, Lessmann, and Schaal 2020), and because they are recurrent it makes the network suitable for processing time-dependencies. However, there is one main reason why the RNN is more likely to perform better and that is because of the LSTM models. These models incorporate memory cells, and in previous research - for example in the research of (Montgomery et al. 2004) - it is shown that exactly those memory cells are the components that make the LSTM perform perfectly when processing sequential data. It is therefore that we expected the LSTM to perform better than the MLP.

The second sub-question to be answered is the following:

> *RQ 2: Does the Feed-Forward Neural Network including product information perform better than the Feed-Forward Neural Network excluding product information?*

Section 4.1 shows that the two MLPs perform similar to each other. This is mainly because of the RAM-memory problem, causing the amount of data to be used relatively small. It is therefore stated that there is no support to claim that the model including product information performs better than the model without product information

The next sub-question to be answered is:

> *RQ 3: Does the Recurrent Neural Network including product information perform better than the Recurrent Neural Network excluding product information?*

As to the performances of the MLPS, the performances of the LSTMs are also similar to each other. Again, this is caused by the running-time crashing because of too little RAM-memory. We therefore cannot say that the LSTM including product information outperformed the LSTM excluding product information.

The last sub-question that needed to be answered is as follows:

> *RQ 4: Which of the four models implemented has the best model performance?*

When looking at the F1-scores and the AUC-scores of the four models, we see that all the performances are equal to each other. This means that there is no evidence to support the claim that one of the four models performed the best, or that a particular sort of neural network outperformed the other neural network. This is contrary to what was expected in this research, and also contrary to what has been shown in previous research.

This leads us to answering the main question, central to this research. Reviewing the results, we argue that the addition of product information does not have an influence on the performance of models predicting purchase behavior, nor does it fluctuate in performances between different kinds of neural networks.

5.2 Findings

The results of this research show that all models perform equally well on the sequential data. As it was expected that the LSTM would outperform the MLP, the results show no evidence for this claim. This is in line with the research of (Benabderrahmane, Mellouli, and Lamolle 2018), who states that MLPs are perfectly suitable for processing sequential data. However, there is also many evidence that RNNs are the most suitable models for sequential data, as they are especially designed for tasks similar to this research. For example, Hochreiter and Schmidhuber (1997) build the LSTM as a solution to the vanishing and exploding gradient problems that occur in simple, recurrent neural networks, making them perfectly capable of processing sequential data with long-term dependencies.

That we cannot support this claim is due to multiple reasons. Because of the RAM-memory problem, the amount of data used in the models is very low and therefore not generalizable. There was too little data to gain valuable information from the results. With a small amount of data it is more difficult for a neural network to find the differences between the two classes that is needed to separate them from each other. As we saw in the results, the models were capable of predicted the majority class but did not succeed in predicting the minority class. The RAM-memory problem was probably caused by the one-hot-encoded product sequences, that ended up as very sparse binary vectors requiring a lot of memory. In the future it might be a better idea to encode the product information with a different method. For example, the data can be encoded by training own embeddings using the Word2Vec algorithm (Church 2017). A disadvantage of this method would be that the embeddings created by Word2Vec allow for a larger model than was initialized in this research, to be able to process the data.

The thing we did learn from looking at product information, is that valuable insights for e-commerce companies can be gained by looking at the distribution of the events that were linked to the products. For example, in Section 3.3 it was seen that a lot of products that were viewed in detail, were also added to the shopping cart, but eventually only a small amount of the added products were bought. This information is related to the principle of *cart abandonment*. When subtracting the removed products and purchased products from the amount of products that were added to the shopping cart, it became clear that a relative big amount of products remained in the shopping cart. According to (Kukar-Kinney and Close 2010), cart abandonment happens when users are eventually not willing or not able to purchase the products they added to the cart, or because, for example, their session broke down. In light of predicting and analyzing purchase behavior, the reasons why products are abandoned in the shopping cart is an interesting topic for future research.

5.3 Limitations and Future Research

Except for the RAM-memory problem and the encoding of the product sequences, there are a few more limitations this research was subjected to. As already indicated in Section 1, this research has a limitation regarding the product information that was used. For this study, we were restricted in using only the available unique product identifiers. This information is of small value, because we cannot say anything about the kind of products users viewed or bought during a session. However, as we saw in Section 3.3, it became clear that the information can give valuable information to a certain level. In future research, it would be interesting to analyze more specified product information. This can be information about the kind of products, the categories, the prices, and so on. (Lukose et al. 2010) showed in their research that the product group is a crucial factor in predicting purchase behavior, and according to Park and Kim (2003), information about the prices of products can also be of great influence in the shopping behavior of online customers.

Besides specified product information, previous research shows that there exist more relationships in purchase behavior in an online shopping environment. Other characteristics that may be of influence can be community membership (Olbrich and Holsing 2011), item characteristics (Ramezani and Yaghmaee 2016) and user characteristics (Lo, F., and Leskovec 2016).

Another suggestion for future research is predicting purchase behavior using user-centric clickstream data, instead of site-centric clickstream data. User-centric data focuses on the entire online activity of a user, whereas site-centric data is focused on the online activities of users on a particular website. Often, before a purchase is done, consumers review multiple websites to seek for the best option that will fulfill their needs. For example, a certain product may be for sale with a discount on one website, but is for sale without discount on another website. According to (Lukose et al. 2010)m reviewing websites is an important indicator of purchase behavior, and product information can of influence when analyzing this.

6. Conclusion

This research focused on adding product information to a model that classifies user sessions as buying or window-shopping sessions. The aim of this study was to check whether product information would provide for a better classification, and if this would fluctuate between different kinds of models. The main thing that is learned from this research, is that when adding product information that is represented as a a sparse vector (i.e., a vector containing a lot of zero values), the model becomes limited in processing a small, not generalizable amount of data.

References

Arora, S. and D. Warrier. 2016. Decoding fashion contexts using word embeddings. *KDD Workshop on Machine Learning meets Fashion*.

Benabderrahmane, S., N. Mellouli, and M. Lamolle. 2018. On the predictive analysis of behavioral massive job data using embedded clustering and deep recurrent neural networks. *Knowledge-Based Systems*, 151:95–113.

Bengio, Y., P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bigon, L., G. Cassani, C. Greco, L. Lacasa, M. Pavoni, A. Polonioli, and J. Tagliabue. 2019. Prediction is very hard, especially about conversion. predicting user purchases from clickstream data in fashion e-commerce. *arXiv preprint arXiv: 1907.00400*.

Boyle, T. 2019. Dealing with imbalanced data. *Towards Data Science*. Retrieved from: https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18.

Brownlee, J. 2016. How to build multi-layer perceptron neural network models with keras. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/.

Brownlee, J. 2017a. How to one hot encode sequence data in python. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/.

Brownlee, J. 2017b. How to use word embedding layers for deep learning with keras. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/.

Brownlee, J. 2018a. Use early stopping to halt the training of neural networks at the right time. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

Brownlee, J. 2018b. When to use mlp, cnn, and rnn neural networks. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/.

Brownlee, J. 2019a. 3 ways to encode categorical variables for deep learning. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/.

Brownlee, J. 2019b. A gentle introduction to imbalanced classification. *Machine Learning Mastery*. Retrieved from: https://machinelearningmastery.com/what-is-imbalanced-classification/.

Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Chawla, N.V., K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. 2002. Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

Chollet, François et al. 2015. Keras. https://keras.io.

Church, K.W. 2017. Word2vec. *Natural Language Engineering*, 23(1):155–162.

Dunford, R., Q. Su, and E. Tamang. 2014. The pareto principle.

Eckhardt, K. 2018. Choosing the right hyperparameters for a simple lstm using keras. *Towards Data Science*. Retrieved from: https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046.

Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep Learning*, volume 1 of *2*. Cambridge: MIT Press.

Harris, Charles R., K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'10, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

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

Hunter, J. D. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Janocha, K. and W.M. Czarnecki. 2017. On loss functions for deep neural networks in classification. *arXiv preprint arXiv: 1702.05659*.

Jenkins, P. 2019. Clickgraph: Web page embedding using clickstream data for multitask learning. *Companion Proceedings of the 2019 World Wide Web Conference*, pages 37–41.

Kingma, D.P. and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv: 1412.6980*.

Koehn, D., S. Lessmann, and M. Schaal. 2020. Predicting online shopping behaviour from clickstream data using deep learning. *Expert Systems with Applications*, 113342.

Kowatsch, T. and W. Maass. 2010. In-store consumer behavior: How mobile recommendation agents influence usage intentions, product purchases, and store preferences. *computers in Human Behavior*, 26(4):697–704.

Kukar-Kinney, M. and A.G. Close. 2010. The determinants of consumers' online shopping cart abandonment. *Journal of the Academy of Marketing Science*, 38(2):240–250.

LeCun, Y., Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature*, 521(7553):436–444.

Lemaître, Guillaume, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

Lo, C., Dan F., and J. Leskovec. 2016. Understanding behaviors that lead to purchasing: A case study of pinterest. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 531–540.

Lukose, R., J. Li, J. Zhou, and S.R.P. Venkata. 2010. Learning user purchase intent from user-centric data. *U.S. Patent Application*, 12(263).

McKinney, W. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Montgomery, A.L., S. Li, K. Srinivasan, and J.C. Liechty. 2004. Modeling online browsing and path analysis using clickstream data. *Marketing Science*, 23(4):579–595.

Müller, B., J. Reinhardt, and M.T. Strickland. 2012. *Neural Networks: An Introduction*. Springer Science Business Media.

Olbrich, R. and C. Holsing. 2011. Modeling consumer purchasing behavior in social shopping communities with clickstream data. *International Journal of Electronic Commerce*, 16(2):15–40.

Padmanabhan, B., Z. Zheng, and S.O. Kimbrough. 2001. Personalization from incomplete data: what you don't know can hurt. *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 154–163.

Pandas Development Team, The. 2020. pandas-dev/pandas: Pandas.

Park, C.H. and Y.G. Kim. 2003. Identifying key factors affecting consumer purchase behavior in an online shopping context. *International Journal of Retail and Distribution Management*.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Plotly, Technologies Inc. 2015. Collaborative data science.

Ramezani, M. and F. Yaghmaee. 2016. A novel video recommendation system based on efficient retrieval of human actions. *Physica A: Statistical Mechanics and its Applications*, 457:607–623.

Sakar, C.O., S.O. Polat, M. Katircioglu, and Y. Kastro. 2019. Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and lstm recurrent neural networks. *Neural Computing and Applications*, 31(10):6893–6908.

Schmidhüber, J. 2015. Deep learning in neural networks: An overview. *Neural Networks: arXiv: 1407.7828*, 6:85–117.

Sharma, S. 2020. Activation functions in neural networks. *Towards Data Science*, 6.

Sheil, H., O. Rana, and R. Reilly. 2018. Predicting purchasing intent: Automatic feature learning using recurrent neural networks. *arXiv preprint arXiv: 1807.08207*.

Shickel, B. and P. Rashidi. 2020. Sequential interpretability: Methods, applications, and future direction for understanding deep learning models in the context of sequential data. *arXiv preprint arXiv: 2004.12524*.

StackExchange. 2017. Multiple hidden layers in neural network diagram. Retrieved from: https://tex.stackexchange.com/questions/362238/multiple-hidden-layers-in-neural-network-diagram.

StackExchange. 2019. How do i draw a simple recurrent neural network with goodfellow's style? Retrieved from: https://tex.stackexchange.com/questions/494139/how-do-i-draw-a-

simple-recurrent-neural-network-with-goodfellows-style.
Tagliabue, J., L. Lacasa, C. Greco, M. Pavoni, and A. Polonioli. 2019. Predicting e-commerce customer conversion from minimal temporal patterns on symbolized clickstream trajectories. *arXiv preprint arXiv: 1907.02797*.
Tellis, G.J. and G.J. Gaeth. 1990. Best value, price-seeking, and price aversion: The impact of information and learning on consumer choices. *Journal of Marketing*, 54(2):34–45.
Toth, A., L. Tan, G. Di Fabbrizio, and A. Datta. 2017. Predicting shopping behavior with mixture of rnns. *eCOM @ SIGIR*.
Wardhani, N.W.S., M.Y. Rochayani, A. Iriany, A.D. Sulistyono, and P. Lestantyo. 2019. Cross-validation metrics for evaluating classification performance on imbalanced data. In *2019 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, pages 14–18, IEEE.
Waskom, M. and the Seaborn Development Team. 2020. mwaskom/seaborn.
Weng, C.G. and J. Poon. 2008. A new evaluation measure for imbalanced datasets. *Proceedings of the 7th Australasian Data Mining Conference*, 87:27–32.
Wu, Z., B.H. Tan, R. Duan, Y. Liu, and R.S. Mong Goh. 2015. Neural modeling of buying behavior for e-commerce from clicking patterns. *Proceedings of the 2015 International ACM Recommender Systems Challenge*, pages 1–4.
Zhao, Y., S. Chu, Y. Zhou, and K. Tu. 2017. Sequence prediction using neural network classifiers. *International Conference on Grammatical Inference*, pages 164–169.

Appendix A:

Package	Description and citation
Pandas	Package designed for manipulation and analysis of data. Citation: Pandas Development Team (2020), McKinney (2010)
NumPy	Package designed for the support of big, multidimensional arrays, together with mathematical functions to process these arrays. Citation: Harris et al. (2020)
Keras	Open-source library that offers an interface for neural networks. Built on top of the machine learning platform TensorFlow. Citation: Chollet et al. (2015)
Scikit Learn	Free software-library for Machine Learning. Disposes classification-, regression- and clustering algorithms. Citation: Buitinck et al. (2013), Pedregosa et al. (2011)
Matplotlib	Library designed for creating static, animated, and interactive visualizations. Citation: Hunter (2007)
Seaborn	Provides a library on top of Matplotlib that offers intelligent choices for drawing attractive and informative statistical graphics. Citation: Waskom and the Seaborn Development Team (2020)
Plotly	Interactive graphing library for Python. Citation: Plotly (2015)
Imblearn	Python package offering re-sampling techniques. Citation: Lemaître, Nogueira, and Aridas (2017)