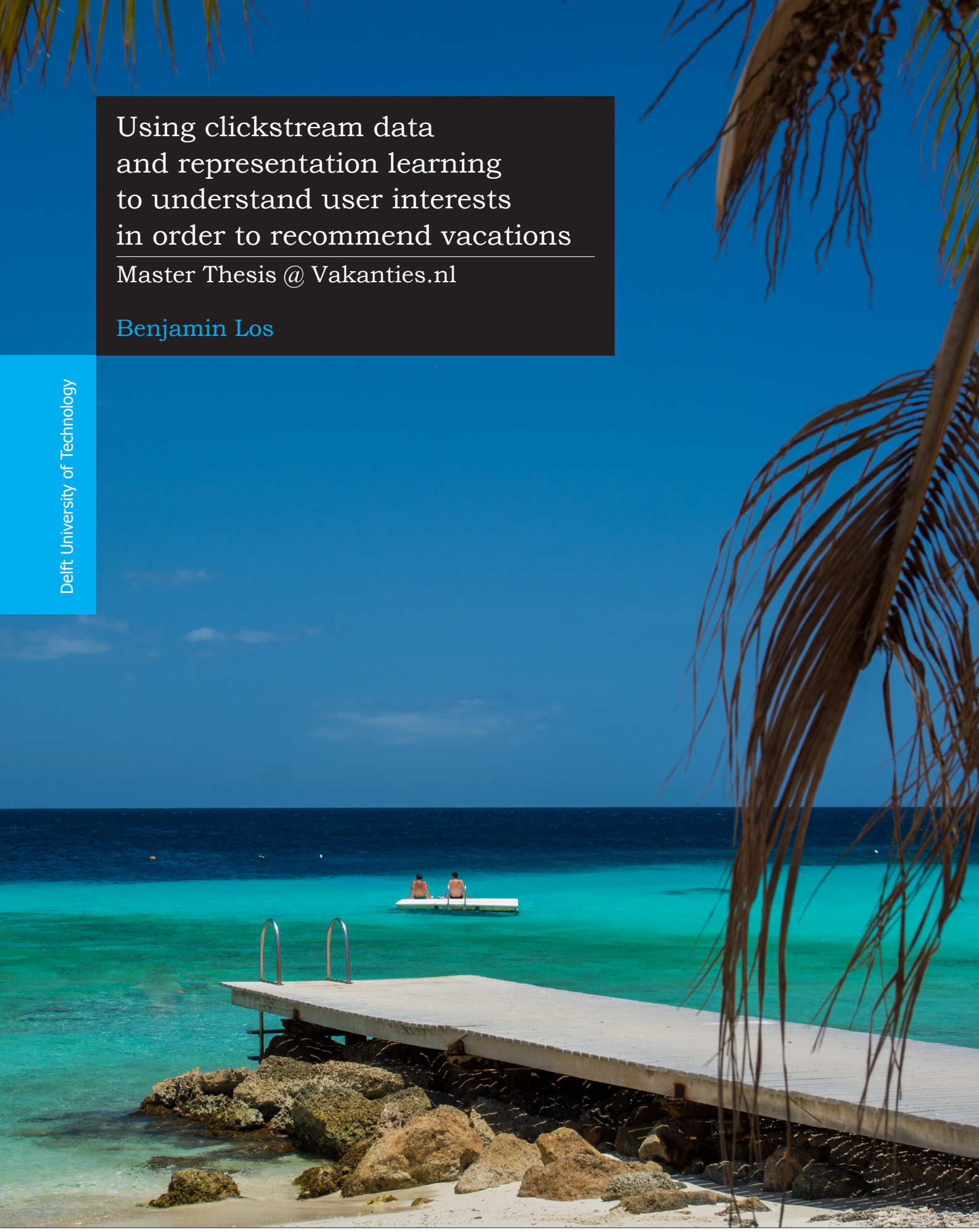
Using clickstream data and representation learning to understand user interests in order to recommend vacations

Master Thesis @ Vakanties.nl

Benjamin Los

Delft University of Technology



TU Delft Delft University of Technology

Challenge the future

Using clickstream data and representation learning to understand user interests in order to recommend vacations

Master Thesis @ Vakanties.nl

by

Benjamin Los

in fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Monday 30th of September at 11h30.

Student number:	4301838	
Thesis committee:	Dr. ir. Neil Yorke-Smith	Supervisor, TU Delft
	Elwin Kamp	Vakanties.nl
	Dr. Claudia Huff	TU Delft

An electronic version of this thesis is available at www.repository.tudelft.nl.

TU Delft Delft University of Technology

Summary

Because of the transfer from brick-and-mortar stores to the web, tourism companies have had an increasing need for good recommendation systems to help the users of their websites find what they want. When developing a recommendation system for tourism, we run into a couple of problems that we would not run into when developing it for e-commerce. One of these problems is the increased effect of the cold start problem. This problem entails that we do not understand what new users are interested in because we have very little information about them. The increased effect of the problem is due to the low number of bookings that are made compared to e-commerce purchases. To reduce the effect of the cold start problem, we can use additional data sources in order to understand the user's interests better.

To simplify the use of the additional data source, we explore the possibility of embedding the data or using it in conjunction with an embedding. Vakanties.nl is a company with the need for an improved recommender system. Therefore, we decided to explore these possibilities in cooperation with Vakanties.nl. We develop a recommender system that is able to make recommendations, using both the embedding and clickstream data from Vakanties.nl. We find that although the results of our system do have potential, the system requires some further improvement to compete with a conventional recommendation system.

Preface

Recommendation systems try to understand what the users want, and to recommend products that they are interested in. However, when you hardly have any information about the user, then this gets pretty difficult. During my thesis I tried to do exactly this for a company in Rotterdam, Vakanties.nl. Vakanties.nl sells customizable vacations to the customer on their website. To help their customers find what they want, they needed a recommendation system. By implementing a recommendation system, I explored the possibilities of using both embeddings and clickstream data to understand the user's interest. I found that, although the results were promising, it still needs further development.

I am very thankful for my thesis supervisor, Dr. Neil Yorke-Smith, who helped me finish this thesis by giving me feedback, guidance, and a mirror to show me the obvious. Mostly, I appreciate the freedom you gave me in finding my own problem, and then guiding me through the process of solving it. I am also very grateful for the help I received from Elwin Kamp, Tim Ebbers, Judith Eyck, and everybody else at Vakanties.nl. Not only did they allow me to use their data, but they also helped me understand the problem and guide me to a solution. Additionally I want to thank Dr. Claudia Hauff for making time to be on the thesis committee. Finally, I would like to express my gratitude to everybody who took the time to give me some feedback on my system. But most importantly, I would like to thank my girlfriend, Amy, for all her help and patience. Behind every great man is a great woman.

Benjamin Los
September 2019

Contents

1

Introduction

Nowadays the travel and tourism world no longer consists of brick and mortar stores. Instead, it is mostly run from servers. The advantage of this is that you do not have to maintain all these locations and that it is available 24/7. The problem on the other hand, is that you no longer have the trusty employees who talk to each customer entering the store. Instead the customers browse your website by themselves, hoping to find something they like. In order to make it easier for the customers to find what they want there are multiple systems; One of which is the recommender system.

Recommender systems are used in various branches of e-commerce, however there are a couple of additional challenges to face when applying it to travel and tourism. In most e-commerce a customer might buy a couple of products per month and will view many more. In travel and tourism, customers will probably only book a single accommodation per year. This introduces a few problems for the recommender system.

1.1. Problem statement

The first problem while using recommender systems in travel is that it has far less information to work with than conventional applications. Some of the most well-known applications of recommender systems are in fields where users have a lot of interactions with the system. For example: the Netflix recommender system[1] has users who, on average, watch more than one and a half hours per day. On the other hand, in 2017 the average Dutch citizen only went on vacation 1.3 times a year[2]; which shows that the field of travel has a strong lack of data compared to Netflix. The lack of data increases the negative effects of the cold start problem. This problem occurs when the system has a new user or product and has to make a recommendation. Seeing as the system does not have information about which products the user is interested in (or which users are interested in the product), the system is unable to make good recommendations.

One way to mitigate the introduction of the cold start problem with new products, is to create an embedding representing the products. An embedding is a low dimensional space that represents a higher dimensional space. In our situation we would like to represent the attributes of the various products in a low dimensional space (e.g. 2 dimensions). Depending on the application, embeddings can have various other desired attributes. An example of this would be our requirement that similar products are close to each other in the embedding.

The second problem with recommendation systems is that it is often a black box system. This means that it is difficult to understand why certain products are recommended to certain customers. Although there are multiple recommender systems that do mitigate this problem, this is still an issue that should be taken into account.

The final issue is that some recommender systems are only able to recommend a fixed number of products. This becomes a problem when, for various possible reasons, you might want to have more

recommendations. One of these reasons might be that some of the products that were recommended are no longer available or that there is enough room on the page to recommend more products.

1.2. Introducing Vakanties.nl

One of the companies that ran into these difficulties was Vakanties.nl; located in Rotterdam. Vakanties.nl is a small company consisting of approximately 20 employees. They sell vacation packages consisting of a hotel, the flights, and sometimes even the transfer between the airport and the hotel. We will be referring to these packages as the products. Vakanties.nl sells their products on the web and allows their users to customize the packages to match the user's requirements. This feature distinguishes them from their competitors.

One of the goals that Vakantie.nl has, is to make it as easy as possible for their customers to find what they want. In order to accomplish this, it is key to understand what the user is interested in. Whether it is a vacation in Spain or Turkey; Whether it is a child friendly vacation or just involves adults; Etc. To understand what the customer is interested in, the company has to analyze the information that they have about the user. Once they have an idea of the users interests, it is possible to make custom recommendations for the user based on their interests.

When designing a recommender system for travel and tourism, there are a couple of difficulties that will have to be faced. For Vakanties.nl this is no exception. Due to the low amount of vacations people take per year, the effect of the cold start problem is fairly big. To minimise this problem, we will have to use additional data about the users. To moderate this on the product side, we will have to recommend products based on the attributes instead of the bookings.

To mitigate the blackbox issue we will have to try to decrease the dimensions of the product attributes to make it easier to understand the relationship between products. And finally, we will have to ensure that we are able to make a varying number of recommendations instead of a fixed amount without having to recreate the recommender system.

1.3. Research questions

The goal of this research is to determine the viability of using an embedding to help minimize the negative effects of the cold start problem for recommender systems. To explore its effects we have constructed the following research questions:

1. How can we create an effective embedding in order to encode the products of Vakanties.nl?
   (a) Which criteria are important for an embedding in order to encode the products of Vakanties.nl?
   (b) How can we encode and decode products to and from the embedding?
2. How can we validate the effectiveness of the embedding for making recommendations?
   (a) How can we use the embedding in combination with clickstream data to learn which products the user is interested in?
   (b) How can we use the interests of the users to make recommendations?
3. How do the recommendations made while using the clickstream data and the embedding compare to the recommendations made by the model currently used by Vakanties.nl?

1.4. Outline

The structure of this thesis is as follows: The following chapter, chapter 2, will discuss the Dutch tourism market, the various variations of recommender systems, and will finish off with introducing some approaches to representation learning. After this, in chapter 3, we will introduce the recommender system we designed and developed. We will talk about each of the components, which make up the system, individually. In chapter 4, we will evaluate the embedding created and the recommendations that are made. Finally we will conclude and discuss some future work in chapter 5.

2

Background

Previously, in chapter 1, we introduced the problem at hand and the goal of our research. In this chapter we would like to give some background information; both about the travel and tourism field in the Netherlands as well as some information about various solutions to the problems we introduced. The chapter is divided into four sections. First, we will introduce the world of tourism, in section 2.1. After this, in section 2.2, we will discuss various approaches to recommender systems. In section 2.3 we will talk about some of the methods that exist to make an embedding (representation learning). Finally, in section 2.4, we will explain some models that we will use in our system.

2.1. The world of tourism

Nowadays tourism is part of e-commerce; there are tons of websites allowing you to buy vacations all over the world, with more possibilities than you could experience in a life time. How this developed from the old brick-and-mortar travel agencies we will discuss in section 2.1.1. Afterwards, in section 2.1.2, we will quickly create an image of what the Dutch tourism industry looks like. In section 2.1.3 we will explain the difference between conventional e-commerce and tourism e-commerce; apparently selling vacations and selling clothes is not the same thing. Finally, at the end of section 2.1.3 in subsection 2.1.3, we will explain why we need recommendation systems in the world of tourism e-commerce.

However, we first need to clarify some terms. Tourism is a broad field; it can include giving tours, rating museums or restaurants, or sharing photos as inspiration for somebody's next trip. Therefore, we need to specify what we are exactly talking about. When mentioning tourism we are talking about flights, museums, accommodations, sightseeing, tours, etc. Tourism e-commerce however, means a specific part of the field. What we mean while discussing the term tourism e-commerce is selling vacations using a website. This almost always includes: flight and accommodation. Furthermore, it often includes the transfer between the accommodation and airport, and service at the accommodation, such as meals. Sightseeing, tours, museums, etc. are not included.

2.1.1. How has computer science changed tourism?

In the 1960s the development of Computer Reservation Systems (CRS) allowed airlines, and later travel agents, to store flight, hotel and car rental data in one system that could be accessed remotely [3]. In the 1970s and 1980s the Computer Reservation Systems evolved to allow travel agents access to the system which is also called a global distribution system (GDS). These systems allowed travel agents to check the availability of flights, hotels, and car rentals and even make bookings.

In the 1990s suppliers were able to distribute their products using a new channel (the internet) thereby making the internet a new distribution channel and a marketing medium. The user benefited from the internet especially since they were able to find information about their destination more easily and independently, therefore making them more knowledgeable. However, this also lowered the entry barrier for new players to enter the tourism market. By the late 1990s multiple online travel agencies had joined the market and started providing direct access to the travel market. This development

decreased the transaction costs, eliminated coordination mechanisms from other distribution channels, and increased volume discounts. These three factors allowed for lower prices, thereby making it very interesting for users. [4, 5]

By the early 2000s almost every tourism company had developed a website; either a relatively simple static website or a more dynamic website that allowed for things like search and reservations. New parties started to join the market that provided new online services such as consumer review websites, whose sole purpose is to collect consumer reviews and rate the different products. Towards the end of the decade new technologies started to emerge. GPS, internet, and cameras became normal technologies to have in the palm of your hand wherever you went. This allowed for users to connect in social communities both at home and while traveling, further changing the travel and tourism domain. Nowadays most tourism and travel companies operate online; brick-and-mortar travel agencies are a thing of the past. [5]

2.1.2. The Dutch tourism e-commerce market

The latest reports on the Dutch tourism market are from 2017, therefore all the numbers used will be based on 2017. Out of the 16.8 million Dutch citizens, 14 million went on at least one vacation in 2017. In total, Dutch citizens went on 18.4 million vacations within the Netherlands and 22 million outside of the Netherlands. This means that the average Dutch citizen went on 1.3 vacations outside of the Netherlands. 40% of these vacations were purchased as a package or composed from various possibilities. The most popular destination was Germany, followed by France, Spain, Belgium, and Italy. 72% of the vacations outside of the Netherlands was booked online.[2, 6]

For a company like Vakanties.nl this means they potentially have 5.6 million customers, seeing as they only sell packaged vacations. However, given that only 72% book online this is reduced to about 4 million. The average customer will book a little over one and a half vacations a year. This means that if customers choose to return to Vakanties.nl again, it will be most likely they only return once a year.

2.1.3. Comparing tourism with conventional e-commerce

While we have shown how the tourism world has developed it seems like it has changed in a comparable way to the rest of the e-commerce world: they sell products online, there is a lot of competition, and the newest techniques in IT are being applied. [4]

Although there might be a lot of similarities between selling books and selling vacations, there are also two key differences: attributes of the product and experience of the product. We will shortly elaborate on both of these differences. Afterwards we will explain the necessity of recommendation systems.

Attributes of the product

A vacation is often a combination of multiple products: a flight to the country of destination, a transfer to the hotel they are staying at, lodging, meals, a transfer back to the airport, and a flight back home. This could become even more complicated with longer vacations, and with multiple destinations. The difficulty arises when each of these products have to fit together: be available at the same time (and/or following each other) and allow for the restrictions of the party traveling (number of travelers, luggage limits, room configuration). Each of the separate products might have pretty clear attributes but when fitting them together and selling them as a single product we do not only have a big amount of attributes but we also have very segregated and complex attributes. We encounter complex attributes like the weather at the destination when visiting or the availability of the vacation (flight, transfer, and lodging), but also very specific attributes that might not be possible for all the destinations that the user could visit. These attributes make it very difficult to handle, because there is no standard format to house the information. [7, 8]

Experience of the product

While predicting the experience of people to a specific product is difficult in every field, it is extra difficult for travel and tourism. When predicting how a person will experience a specific hammer, we can look at what other people thought about the hammer. Some people might disagree but if the

hammer is well made most people will like it and if it is badly made most people will dislike it. When we try to predict how people experience a book, we cannot just look at how other people experienced it; different people like different kinds of books. We can, however, look at users who are similar to the person in question. If those people enjoyed the book this person will probably do so as well. [9]

Unfortunately, neither of these approaches work if it comes to vacations. A trip to a sunny beach will probably be enjoyable for someone who adores the sun, but it might not be for the adventurous type. However, when we look at similar users we also encounter a problem. First of all, not many users go on exactly the same vacation. They might go to the same hotel but in a different season; using a different flight; or go a few years later when, possibly, the hotel is not as glamorous as it once was. Contributing to the problem is that people normally only have one or two vacations a year while they might read five or even ten books (movies will most likely even be more). To make matters even worse, the vacation might not even be the same. Two users who bought exactly the same vacation might have very different experiences; not because of their personalities but because one might get the flu while the other visits beautiful sights; one might get pick-pocketed while the other might find the love of their life. [7, 10]

Why does tourism e-commerce need recommender systems?

In the previous sections we have shown that although tourism e-commerce looks a lot like conventional e-commerce, there are some important differences. Nevertheless, we have also seen that the products have many attributes and that they can be configured in many different ways, essentially giving a lot of different products with many different attributes. These two factors make tourism e-commerce an information and search problem. Together with the fact that the experience of a product is difficult to predict, this makes it a key candidate for recommendation systems. By allowing a recommendation system to recommend vacations to the user, it should allow the user to find what they want more easily and prevent the user from getting overwhelmed by the vast quantity of travel information. The recommendation system is essentially working as a filtering tool. [10, 11]

2.2. Making Recommendations

We now know that tourism e-commerce is complex and that the users have a lot of information to process. Therefore, they might appreciate a recommendation system. First, we must understand what a recommendation system is. We will explain this in section 2.2.1, together with why we need it. In section 2.2.2, we explain how a recommendation system works. And finally in section 2.2.3, we will explain how we can determine whether our recommendation works the way we want it to work.

2.2.1. What are recommendations and why do we need it?

A recommendation is essentially an advice or a proposal: a product that we think the user will find interesting. By making recommendations we allow the users to look at our recommendations instead of all the other options, allowing the users to hopefully find what they are looking for more quickly. To achieve this we must filter out all the irrelevant products and only recommend the most relevant product. Although a hammer might be better than another hammer based on its attributes, many products do not have a clear superior or inferior version, since it depends on the user's preference. Therefore, to make a recommendation we do not only need to understand the products, we also need to understand the preference of the user. [12]

The different systems which make recommendations, also known as recommendation systems, are defined based on how they try to determine the preference of the user. There are three main categories: collaborative, content-based, and hybrid filtering.

- Collaborative filtering uses similarities between users. If a user went on a vacation in Spain and the system knows that many users who went on vacation in Spain also go on Vacation in Italy, then the system should also recommend the user vacations in Italy.

- Content-based filtering uses similarities between items. If a certain user always goes on vacation to the beach, then the system should probably recommend another vacation to the beach.

- Hybrid filtering uses both content-based and collaborative filtering. By combining the two examples shown above the system might conclude that it should recommend a vacation to a beach in Italy.

These are the three main categories. However, there are many more types of recommendation systems which we will not discuss further in this paper. [9]

2.2.2. How can we make recommendations for tourism?

To determine which recommendation systems are most appropriate for tourism e-commerce we must first analyze which data we have available. Without any information about the products or the users we are still able to apply a very basic form of collaborative filtering. A website in the tourism e-commerce, however, must have information about the products it sells. For example: the availability of several hotels; the cost of different transfers; and the times a flight departs. Using this information alone we are able to apply content-based filtering, given that the user has booked a vacation before. In the case that a user has not yet booked any vacation with the system before, this is the cold-start problem[13]. When we have no knowledge about a user we cannot make any recommendations because we do not know their preferences, the same holds for a new product. Both collaborative and content-based recommendation systems have trouble with this. In tourism e-commerce this is a bigger problem than for conventional e-commerce mainly because users only book one or two vacations per year, thereby giving the systems less information to work with. [14]

Another source of data is click-stream data. This is data from the users who are visiting the website and might not have booked a vacation (yet). From this information we could potentially understand which products are similar but also what these users are interested in. If a user is looking at vacations in France, then this is probably what he is interested in. Recommendation systems that use this sort of information are called knowledge-based recommendation systems. Knowledge-based filtering can be used on its own [15, 16] or can be combined with other filtering techniques. When combining this with content-based filtering to form a hybrid recommendation system, the system is able to understand which products the user is interested in but does not meet its requirements (products that are viewed but not booked) using knowledge-based filtering. The system is then able to find similar items that might satisfy its requirements using content-based filtering. A hybrid system like this will have less problems with new users because the system gets information when the user views their first product. A new product could still be a problem for the system [14].

2.2.3. How can we evaluate the quality of a recommendation?

To evaluate the recommendations we have two main approaches: theoretical or practical. Theoretical approaches look at past information of what users chose to buy/book and what the system knows about the users and the products. It then uses this information to make a recommendation for a user before it compares this with the actual booking/purchase (classification accuracy). After this it can be represented using a precision-recall curve or a f1 score. We could also look further then just correctly classified (1) or misclassified (0), which is a binary scoring metric. It would be possible to look at distance metrics if we have a misclassification. Instead of saying ”we have a total fail”, we might be interested to know if the products are similar. If the system recommends a certain shampoo and the user does buy a shampoo -just a different kind- , then this is more accurate than when the user would buy a bouncy castle.[17]

With a practical evaluation the user will make a recommendation to an actual user. This means that the system does not yet know what the user is going to purchase/book. This introduces a couple of challenges but also some advantages.

1. First, not knowing the true interest of the user (what and if they are going to buy), makes it challenging to rate the recommendation. The simple fact that we gave the user a recommendation could have persuaded the user to buy a product instead of leaving empty handed or vice versa. It could also have made the user buy a different product. We therefore cannot simply calculate the distance between the recommendation and the purchase of the user.
2. Secondly, we do not only have users who intend to actually make a purchase. In the theoretical approach we can filter out the users who do not make a purchase allowing us to compare the

remaining group's purchases with the recommendation. With a practical approach we do not know beforehand which users are looking to make a purchase and who are just browsing. This makes it difficult to differentiate between users who do not intend to purchase anything and users who could not find what they were looking for but are willing to make a purchase.

3. The practical approach does require a full working system, while the theoretical approach will work as long as you have the data and are able to make a recommendation. This also means that if the recommendation system is giving bad recommendations, this could result in missed purchases.

4. Giving a recommendation might change the outcome of the user's visit. With the theoretical approach we do not account for this, therefore the outcome of the theoretical evaluation might not coincide with an actual visit. The practical approach does not have this problem because it is based on real visits.

Evaluations based on a practical approach are more challenging, however they should also be more accurate.[18]

With practical approaches we have a couple more metrics besides classification accuracy that we can use to gain insight into the system. Because it is a practical approach evaluated in a real system we are able to look at information like click rate, to see how often recommendations are clicked on. This could give insights into the interest of the user and how the user is effected by the recommendation. Another such metric is the conversion rate, which is the percentage of users who eventually make a purchase. If this is a large percentage of the users this could be a result of good recommendations.

Although these metrics do have a meaning by themselves, they are not very useful to evaluate a system unless we are able to compare them against something else. To achieve this there are two methods that we will discuss: multiple recommendations and a/b testing. Multiple recommendation means that each user of the site will be given multiple recommendations originating from different recommendation systems. When the recommendations of a certain recommendation system achieves better metrics than the others this could indicate that this specific recommendation system is better than its counterparts. However, this could also be attributed to other factors, such as: placement, order, loading time, visual attributes, etc. This means that when implementing such comparisons it is important to make these recommendations as similar as possible in all aspects except for the actual recommendation.

A/B testing does not compare recommendations next to each other but compares them by providing them to different users. A user will be provided with one of two (or more) variants of the website. These variants should be identical in all aspects except for one. In our case this would be the recommendations. By taking the comparison to a broader level by differentiating between users instead of recommendations, some of the factors in which recommendations could still vary with the multiple recommendations approach, are no longer an issue. Placement, for example, should no longer vary between variants of the system, while this was still an issue with the multiple recommendation approach.[17]

Overall it is important to understand the difference between theoretical and practical evaluation approaches. The theoretical approach might make sense when still developing the system, however this has a risk of being incorrect when comparing it to reality. Therefore, it is important to use a practical approach to evaluate a system when it nears the end of development.

2.3. Representation Learning

As explained in the previous section, recommendation systems try to predict what the user is interested in. These systems are able to use various kinds of data to accomplish this. Be that as it may, if the data is too complex the system will be unable to accurately recommend the user. To solve this we will look into representation learning. First we will explain what representation learning is. We will then continue to explain multi-view representation learning. Finally, we will look into the advantages of

applying representation learning. How we will incorporate the embedding and our specific requirements will be discussed in chapter 3.

2.3.1. What is representation learning?

Representation learning, also known as feature learning, are techniques that aim to try and find a new set of features, also known as a representation or an embedding. The embedding should make a certain task easier compared to using the original features. Classification is an example of such a task, by using an embedding it allows the model performing the classification to access the information more easily. Representation learning has managed to accomplish impressive results, with recent developments. This is often achieved with complex models, some of which we will explain in 2.4.

There are various methods to learn an embedding. A couple of the more straight forward methods involve clustering techniques or Principle Component Analysis. However the more complex or deep methods, with multiple layers, will often use some type of Restricted Boltzmann Machine or Autoencoder. The method depends on the data used and the requirements of the embedding.[19]

2.3.2. What is multi-view representation learning

In our case we would like to find a representation that makes it easier to generate recommendation from the data. The data that is available for us is: click-stream data from the users visiting the website and categorical data about the products that Vakanties.nl offers. From these two datasets we would like to generate recommendations. To help this process we would like to create a representation that makes it easier to make recommendations. This process, of combining multiple datasets into a single representation, is called multi-view representation learning.

There are two categories of multi-view representation learning: *Multi-view representation alignment* and *Multi-view representation fusion*. Multi view representation alignment tries to align the two (or more) original datasets. This requires some type of distance-, similarity- or correlation metric to make the alignment. Once we have aligned the datasets, the elements of one dataset will correspond with certain elements in the other dataset. For example we could align our two datasets in a 1-to-many relationship, where each accommodation will correspond to the 0 or more click-streams that end up booking that accommodation.

Multi-view representation fusion uses the datasets to create an embedding in which the data is represented in such a way that it is comprehensively represented for the given task. This can be accomplished using several different techniques. These techniques can be separated into two subcategories: *Graphical model-based fusion* and *Neural Network-based fusion*. Graphical model-based fusion tries to learn a probabilistic model that fits the distribution of the embedding best in order to represent the information best for the given task. Neural Network-based fusion tries to learn the embedding using deep learning techniques for representation learning that are adjusted for multi-view. An example of such a technique is a multimodal deep autoencoder. [20, 21]

2.3.3. What are the advantages of applying representation learning?

By applying representation learning to create an embedding of the original data in a lower dimension we are simplifying the representation of the items in the embedding. This has two main advantages:

- The first advantage is that because of the simplistic representation it is often easier to use by various algorithms. For example: a neural network would require less neurons to encode an item of an embedding than the original representation. This allows the network to have less in its input/output layer, making it easier to train.

- The second advantage is that humans have difficulties understanding high dimensional spaces. By learning an embedding in a lower dimension it is made easier to comprehend what the relationship between various items in the embedding are. Of course this does depend on the quality of the embedding. If the embedding does not represent the items well, this could mean that the relationship between items translates to the embedding poorly.

2.4. Machine learning models

In our system we will experiment with various types of machine learning models. These models will in general try to make a prediction given the data that they have seen before. The main idea behind these models is that given enough data they will start to understand the patterns in the data. When then model encounters new data it should still be able to make it predictions given that it understands the patterns properly and that the new data adheres to the same patterns.

In this section we will discuss three types of models: The first models are Artificial Neural Networks, these are the simplest form of neural networks, which we will discuss in 2.4.1. Then, in section 2.4.2, we will introduce a variant made for sequential data, the Recurrent Neural Network. In section 2.4.3 we will introduce our last model, the Convolutional Neural Network. Finally in section 2.4.4 we will explain a bit about training these models and how we can further improve the training.

2.4.1. Artificial neural networks

The first model that we will discuss is the Artificial Neural Network (ANN), depicted in figure 2.1. As the name suggest, this is a network made out of artificial neurons. These neurons are positioned in layers, where a layer could have anywhere from 10 to 1000 neurons. These layers are then stacked to create the network of the ANN. Each neuron receives the output from the previous layer and multiplies this with its weights, adds it together and passes this value through a activation function in order to generate an output. An ANN receives the input on one side, sends it through the layers, and out comes the prediction on the other side. To train an ANN you need an error, the error is then sent back through the layers in reverse order, each layer using the error as feedback to know how it should change the weights of its neurons to improve the prediction. This feedback method is know as back propagation.

The number of weights that a model has is an indication for how hard it is going to be to train the model. Seeing as each weight has to be trained, the more weights the harder it is. The number of weights that a layer has is equal to the number of neurons of the layer times the number of neurons in the previous layer plus one. The multiplication is because the layers are fully connected, each neuron receives the output from each neuron in the previous layer. The plus one is because each neuron also has a basis weight. This weight is subtracted from the sum of the weight times the outputs of the previous layer. All in all this means that the number of weights that a network has can grow quite fast.

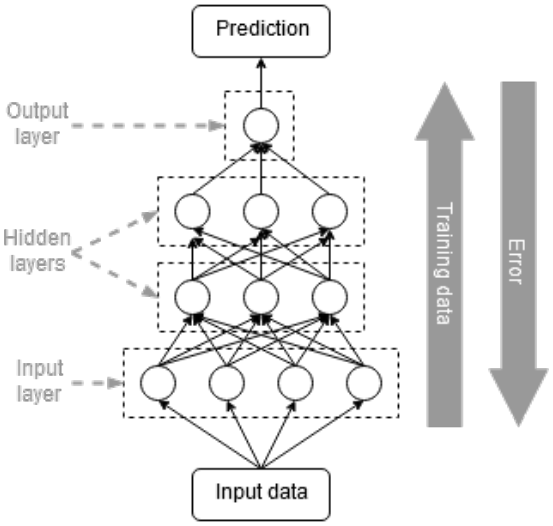


Figure 2.1: Depiction of the structure and the components of an Artificial Neural Network (ANN).

The origin of the ANN is not that straight forward. The various ideas/methods that are required for the network were created by different people. The idea of modeling a neural network was first published in 1943 by W.S. McCulloch and W. Pitts[22]. The perceptron (a.k.w. Artificial Neuron) was created by F. Rosenblatt in 1958[23]. And there were many others responsible for developing the various methods and ideas that the ANN is based on.

2.4.2. Recurrent neural networks

A RNN is an Artificial Neural Network that excepts sequential data and feeds the layers outputs to itself. The model will receive the first input from the sequence on time step 1, the second on time step 2, etc. The model will feed the layer outputs that it has on time step t to the same layer on time step

$t + 1$. The model therefore has access to all the input from the previous time steps, some directly and some indirectly.[24]

One major problem that RNNs run into is the vanishing gradient. Once the sequence gets very long, the feedback/error that the model receives at the end of the sequence works fine for the later time steps. However, it is not strong enough to reach the model at the lower time steps. It seems to vanish, hence the name vanishing gradient.

A model that tries to mitigate the effects of the vanishing gradient problem, is the Long short-term memory (LSTM) model. By adding additional units to the model, that are able to store the gradient for longer amounts of time, they lessen the effect of the vanishing gradient. These units can store a value over multiple time steps, only releasing it when the model gives it the signal to reveal the value. The feedback follows the exact same root back, therefore allowing it to skip multiple time steps and therefore keeping it potent.[25]

The gated recurrent units (GRUs)[26] is a simplified version of the LSTM. While an LSTM unit has three gates (input gate, output gate and a forget gate), a GRU only has two (update gate and reset gate). By reducing the number of gates the unit reduces the number of weights that have to be trained. It therefore makes it performance wise easier to train.

2.4.3. Convolutional neural networks

When a Neural Network has trouble gasping a complex pattern/problem, one of the common ways to help the network out is by increasing the number of layers. The data then has to travel through more layers, allowing each layer to solve a small part of the problem. Oversimplified this would mean that networks with more layers are able to tackle more complex problems. When we apply this concept to ANNs, we run into a problem. Because the layers are fully connected, meaning each neuron receives *all* the outputs of the previous layer, ANNs have a lot of different weights. This poses two problems: First, all the weights need to be trained. The more weights that a model has, the harder it is to train. The second problem is that because of all these weights, it becomes very easy for the model to not learn the pattern of the data but just the desired output. This is know as overfitting. When a model overfits, it has remembered the outputs that it has to give for the input data. The problem arises when the model receives unseen data, seeing as it does not know the desired output the accuracy of the prediction will be bad compared to training.

Convolutional Neural Networks (CNNs) try to fix these problems by limiting the number of weights it has. It achieves this by not using fully connected layers, instead each neuron will just receive the outputs of a part of the neurons on the previous layer, this is known as a convolution. Normally the convolution is a certain section of the neurons in the previous layer, also known as its receptive field. By giving each neuron a different section/receptive field of the previous layer, the current layer will still receives all the information while the individual neurons do not.

CNNs have another smart trick up their sleeve. To further reduce the number of weights, they share them among the layer. A set of weight to cover a section or receptive field is called a filter. These filters can then be shared among the entire layer. CNNs often have multiple filters per layer, this means that they also output multiple value per neuron. This makes the output two dimensional instead of single dimensional.

To counteract the extra dimensional output a pooling layer can be used after a convolutional layer. A pooling layer combines multiple values, for example by taking the average or the maximum value. Their are two variants of pooling layers: global and general. Global pooling layers reduce a entire dimension of the data. This can be used to bring the two dimensional data back to a single dimension. General pooling layers just reduce the length of a dimension by applying its function (mean, max, etc.) to a part of the data similar to a convolution.

CNNs can be used on various dimensional data. If for example we would use a CNN on two dimensional data the convolution would use a square of the previous layer's output instead of a line.

2.4.4. Improve training

Sometimes when training a model will overfit. This can be rather difficult because we still want to get accurate predictions in production. Often a different model would be less susceptible for overfitting for this specific problem. But sometimes this does not help or it is not enough. The second thing that you can try is applying regularization. There are two variants, L2 regularization and dropout. L2 regularization adds a term to the cost function that we are trying to minimize using backpropagation. This term is the L2 length of the weights, this discourages large weights which our often responsible for overfitting. Dropout is a process that randomly drops certain values based on a given probability. This can for example be applied between layers, which will cause some of the outputs of the previous layer to be dropped instead of being passed on to the following layer. This motivates the network to not rely on certain values, which in tern discourages the model to give those values large weights. If this also does not work, the model might just need additional training data.

3

System

Now that we understand something about the Dutch tourism industry, as well as recommender systems, representation learning, and some models, it is time to explain our system. In this chapter we will talk about the various components that make up our system. We will also mention some previous attempts as well as some difficulties we ran into along the way.

We will start, in section 3.1, with the prepossessing steps we have to apply to the data for the various models. Then, in section 3.2, we will discuss our first system design and talk about why we did not continue it. After this, in section 3.3, we will explain how we create our embedding. In section 3.4 we will discuss how we encode user interest. And finally we will introduce the components that make the actual predictions, in section 3.5.

3.1. Preparing the data

To make recommendations, we first need to train the models in the system. And in order to train these models, we need to provide them with data. Vakanties.nl has two kinds of data that are interesting to our use case: Product data, which contains attributes about each of the products that Vakanties.nl offers; And clickstream data, that has information about how the users interacted with the website.

Before we can use the data to train the models we must first ensure that the data is in the right form so that the models are able to understand it. This means the we first have to process the data, before providing it to the models. The processing that is needed for the datasets varies between the two datasets, we will therefore discuss each separately.

3.1.1. Product data

The product data is the dataset containing the various attributes that apply to the hotels. Examples of these attributes might be: The country that the hotel resides in; The numbers of stars that the hotel has on a reviewing site; Or the distance from the hotel to the beach. By analyzing these attributes we would like to understand the products that Vakanties.nl has available.

Before we are able to provide this data to the model, we must first combine the data into a single dataset. Then we must remove the faulty data and the attributes that are of no use to us. After this we are finally able to write it to a format that the model can use. We will shortly discuss each of these steps in addition to a separate step needed for the clickstream data.

Combine the data

Vakantie.nl has two sources of product data, an up to date version of their current products and an outdated version containing some additional attributes. Seeing as we are using historic data, we have products that are exclusive to both lists. Therefore it was important for us to combine the two lists. This added a couple of problems though.

Both datasets have their own set of attributes/features, these features overlaps partially meaning that some features are exclusive and some are shared between the datasets. To make sure we had all the data available, we decided to combine it as much as possible. We managed to transfer almost all of the features, and only had to leave out some features that were contradictory.

Cleaning the data

After merging the two datasets into a single dataset, we started to cleanup the data. Some of the features only had values for a small subset of the products, while others were just unnecessary for our problem. The phone number of the hotel is an example of a feature that we decided to exclude because we did not need it. We further decided to remove strongly lopsided features. These are features that almost always have the same value or are at least very similar.

We further decided to exclude the region, sub-region, and city as these were difficult to encode in an effective matter. The problem with these features is that they are categorical features with very many categories. If we would choose to include them we would have to include a separate feature for each category, making the encoding very large.

Encoding the data

Once we ensured that we had a single dataset containing the information that we wanted, we started encoding the data. By encoding the data we translate the data into a form that is understandable by the models. Each hotel has a feature describing the country that it resides in. This is a simple country code, ES for Spain and NL for the Netherlands. This poses a problem because models only except numbers as input. We therefore have to translate these country codes into numbers. To do this we used one-hot encoding, this creates a single feature for each country code. We can then simply use binary to explain which country it resides in, a 1 in ES and 0 everywhere else means that it resides in Spain. We used one-hot encoding for all categorical features.

The numerical values also posed a problem. One of the features is the number of rooms that a hotel has. The problem is that this value can be quite high, making the model focus more on the feature with high values than the rest. To mitigate this we applied min-max scaling to all numerical features. This ensures that 0 is the lowest value for each feature and 1 the highest. Because all the values are now between 0 and 1 the model should not be forced to focus on one feature more than the others.

Creating path names

Except for preparing the product data for the models we have one additional task that has to do with the clickstream data. The clickstream data consists out of various events that happen on the website of Vakanties.nl. Some of these events are affiliated to a page of a hotel. For example, clicking on an image of the hotel creates an event, this event was created of the page of the hotel. Therefore it might be useful for the models that use the clickstream data to have access to the data of the hotel. In order to connect the event to the hotel id we must generate the Uniform Resource Identifier (URI), seeing as some events do not have access to the hotel id but just the URI of the page.

To generate the URI we used three of the hotel features: the country, region, and city. Before we could concatenate them to form the URI, we first had to remove some of the punctuation marks. We then stored our generated URIs together with the hotel ids in a lookup table to allow the events to be linked to the hotels.

3.1.2. Clickstream data

The clickstream data encapsulates the activities of the users on the website of Vakanties.nl. Whenever a user performs a certain activity, the website will create an event that then gets stored. These events contain information such as: the time, the user, the activity, the web page, the IP address, etc. A session is a period in which the user is active on the website. Events therefore contain a lot of information about the user's behaviour and might be useful in order to understand what they should be recommended.

Another key requirement for the models is feedback. As explained in section 2.4.1, a model needs to have examples of the correct answer in order to train. In our case the models require examples

of good recommendations, the best recommendation would be the product that the user eventually booked/bought. We therefore only require the events that belong to a session that contains a reservation of a hotel.

In order to serve the data to the models, we must first process it to ensure that it is in the form that the models require and that it contains all the information that we think might be useful. There is however an additional step required that was not required for the product data. In order for the models to understand the user's behaviour, they must have access to the entire session. To do this we need to group the events based on the user involved.

The naive approach to grouping the events would be to go over each event and put the events in piles according to the user. The problem with this is that it requires us to keep all the events in memory or to save them in a separate file as groups. We are required to do this because we do not know when a group is complete unless we have visited all the events. The second problem is that once we grouped them we would throw away all the groups that do not contain a reservation. We are unable to determine which groups have a reservation during grouping because the reservations could happen in a later or earlier event. To solve this we will pass over the data two times. We will discus each pass separately.

First pass

During the first pass we will perform two tasks that will make the grouping and filtering in the second pass easier. In order to simplify the filtering we will record the users that make the reservation. By recording this information we can simply filter out all the events that do not correspond to one of these users. Unfortunately, this does not fix the grouping problem. Even though we are able to filter out a bunch of the events, we still need to pass over all the events before we are sure that we have all the events that belong to a certain group. To solve this problem we can count the number of events that belong to each group/session. Seeing as we already have to make an additional pass over the data, this seems to be a viable solution. When grouping we can now start to process a session once it contains the number of events that we counted during this step.

Second pass

During the second pass we first filter out the events not containing a reservation and group the events according to the user to make the session. We then need to process the sessions so that they contain all the information that we want and then encode them to make it comprehensible for the models.

During the processing step we rewrite some of the current information and add new information. For example: we will write the timestamp as two floating numbers, one specifying the time of day, while the other specifies the time of year. However, we will also add the hotel data corresponding to the event as explained in section 3.1.1. We also add the labels to the sessions. The labels are the feedback/examples that the models require in order to train. In our case these are the products that the user books during the session.

After processing it is time to encode the session. For this we encode each event separately similar to the way that we encode the products in section 3.1.1. We also encode the labels, the label weights, and the user id separately. These three in combination with the list of events represent a session and will provide the information to the models. We then combine the encoded features, serialize them and write them to a TFRecord file. TFecord files are a special type of files that TensorFlow[27] models use. It allows the models to read the data linearly from the files instead of having to read the entire file at once.

3.2. Combined approach

The first approach we tried was the combined approach, which was based on multi-view representation fusion. In this section we will explain how the system was constructed; how this system got trained. And finally why we chose not to use this system and instead design a different approach. We will talk about each of these subjects in the subsections 3.2.1, 3.2.2, and 3.2.3 respectively.

3.2.1. Structure of the combined approach

Our first design of the system is based on multi-view representation fusion as described in section 2.3.2. It uses both the clickstream data and the product data and consists out of two encoder models and a decoder model as depicted in figure 3.1. The models *Product Encoder* and *Product Decoder* are both implemented by a multi-layered neural network. Together they form a conventional autoencoder, which encodes the product data into the embedding. It further has the *CS Encoder* which encodes the clickstream data into the embedding. This model is implemented using a RNN (Recurrent Neural Network), seeing as it has to process sequence data from varying lengths.

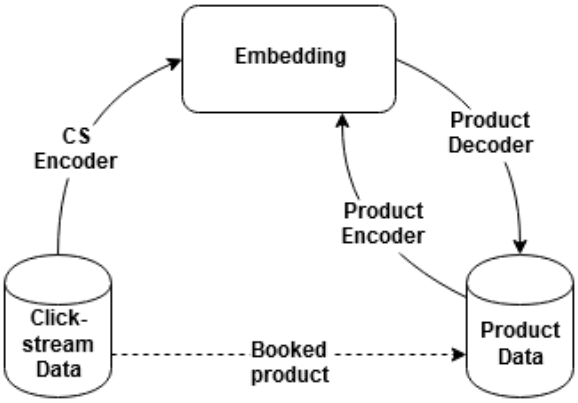


Figure 3.1: Shows the structure of the combined approach. It uses the clickstream data and the product data to learn an embedding. It consists out of three models depicted by solid arrows. The dotted arrow depicts a relationship between the two datasets.

To train the models it is key that we are able to generate an error as feedback to the model. To do this, we require two items from the same data source to compare. By creating a path along some of the models that starts and ends at the same data source, we are able to generate feedback in the form of an error. We call this a training loop, of which the system has three:

- It can be trained using the conventional autoencoder (*Product Encoder* and *Product Decoder*). The *Product Encoder* will receive a product which it will then encode. The *Product Decoder* will try to decode the product, which should result in the same product. To train this model we will use the city-block distance as error.
- It can be trained using *CS Encoder* and *Product Decoder*. Here the RNN will be fed with a clickstream session which it will then encode into the embedding. The *Product Decoder* will at that time be used to decode the encoded clickstream session which will then have to correspond to the product that was booked during the clickstream session. If it does not correspond the error is equal to the city-block distance between these products.
- It can also be trained using the two encoders: (*CS Encoder* and *Product Encoder*). By encoding both the clickstream session and the corresponding product using *CS Encoder* and *Product Encoder* respectively, we are able to obtain two encoded items in the embedding. By comparing these two items (L2 distance), we are once more able to generate an error which can be used to train the models.

3.2.2. Training the models

In general there are two different approaches to training the three different training loops: training them sequentially and training them together. We will first go into training the training loops separately and then go into training them together. Before we do so, it must be said that all three loops are necessary, as each training loop ensures slightly different attributes that are important to the embedding.

- The autoencoder training loop ensures that the decoded products are the same as the ones that get encoded. By training this we improve the understanding of the model of how to decode products without introducing it to mistakes made by the *CS Encoder* model.
- The *CS Encoder* and *Product Decoder* loop ensures the quality of the *CS Encoder*, as this is the main loop to train the CS encoder. This is also the loop that will be used in production, seeing as it goes from the clickstream to a predicted product.
- The last loop, *CS Encoder* and *Product Encoder*, ensures that the encoded clickstream is actually close to the encoded corresponding product. This is the only loop that uses the distance in the

encoding as an error. Therefore, it is the only loop training the models to have corresponding items close to each other.

By using all three training loops instead of just one or two, we are able to more reliably predict the corresponding products.

To train the training loops, we will have to feed a loop with the data source required and compare the output to the input we provided. For example, if we would like to train the autoencoder loop we will feed the auto encoder with some products. The loop will then encode them into the embedding and continue to decode them back to products. At that point, we are able to compare the generated products to the products that we provided as input. This way we are able to calculate the error, which will then function as feedback to the training loop, thereby improving both the models in the training loop.

We just described how we could train a single training loop. The problem, however, is that the third loop, for our example the *CS Encoder*, will not be trained in this process. To train all three models we could take turns training a certain training loop. We will then sequentially iterate over the different training loops and train each of them for a certain amount of time (epochs). One of the difficulties with this approach is that when a certain loop is being trained, it is often reversing the work done by another training loop. In other words, training a certain loop untrains the other loops. This does not make it impossible to train although it does make it difficult and slightly tedious.

The second way to train the system is to train the three loops at the same time. In order to train these three loops simultaneously, we will have to somehow combine the three errors into a single error. Seeing as each model collides with at least one model. One of the most straightforward ways of accomplishing this, is to sum the errors together. However, seeing as the three errors might have different priorities we have to add weights before summing them together. An additional benefit of using weights, is that we are able to bring the errors to the same scale. The *CS Encoder* and *Product Encoder* loop for example, uses a distance in the embedding instead of between two products. By using a weight we are able to ensure that the errors are of similar magnitude, allowing all three loops to be trained 'equally'.

3.2.3. Problems and difficulties

Some of the problems with this system have already been mentioned shortly before. However, we would like to discuss the problem here in order to make it clear why we chose to abandon this approach.

First off, the model consist out of three models, *CS Encoder*, *Product Encoder*, and *Product Decoder*. Two of which are used in production (*CS Encoder* and *Product Decoder*), while the third is just used to create a good embedding. This means that a lot of time is spent designing, fine-tuning, and training a model that will not be used after training. This also introduces additional difficulties.

By having to train three models, we also have to have three training loops. Which introduces us to some difficulties during training. First, as discussed before, it is rather tedious to train the loops sequentially, seeing as they will undo the training performed before. Training them simultaneously also does not seem to be as straightforward as it looks. Seeing as the errors are generated using distances in different spaces, the distances seem to be from different magnitudes. We are of course able to mitigate this using the weights, however this does require a additional fine-tuning.

The final and most severe problem that this system faces, is that it is only able to recommend a single product. Often, recommender systems would like to have various products that both explore and exploit the search space. By only having a single product we are not able to fulfill this requirement. One possibility would be to use the *Product Decoder* to generate multiple products from the space surrounding the encoded clickstream data, instead of just the encoded clickstream data itself. The problem with this is that it will only fulfill the exploitation half of the requirement and not the exploration.

3.2.4. How we got to the new design

One of the ways in which we could mitigate some of the problems, is by removing one of the models. In the current system this is not possible as described in section 3.2.2, each model contributes something to the quality of the embedding.

One of the ways in which we could mitigate some of the problems, is by changing the relationship between the clickstream data and the embedding. Instead of encoding a clickstream session into a single point of interest, we could generate an interest score/probability for the entire embedding. A kind of heatmap depicting the interest of the user for every part of the embedding. This would allow us to remove the *CS Encoder* from the system and just keep the two models that form the autoencoder. This would result in the system depicted in figure 3.2.

We will further discuss the autoencoder (*Product Encoder* and *Product Decoder*) and the embedding in the following section, section 3.3. The translation from clickstream session to heatmap will require a separate model. This model will be described after the embedding, in section 3.4.

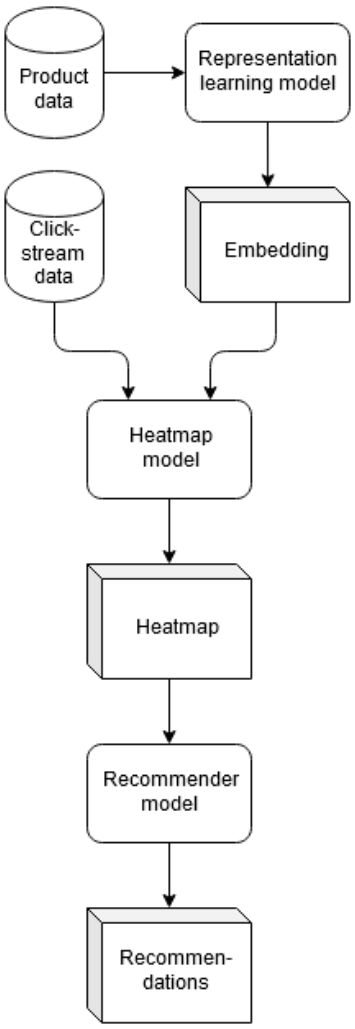


Figure 3.2: Depicts the new system. The cylinders depict the data sources, the boxes represent results from subsystems, and the rounded rectangles depict the models.

3.3. Making an embedding

An embedding is a low dimensional space of a higher dimensional data. The process of creating an embedding is called representation learning. In our system we chose to use an embedding to encode the product data. Each product has hundreds of features, if we were to use the original product space we would have a very high dimensional space. The problem with this is that it is difficult for humans to understand and models require more resources to process it. By using an embedding we do not only make it easier to understand, but we also ensure that each point in the embedding corresponds with a product (whether it exists or not).

In this section we will discuss what makes good embedding (requirements) and how we can make them. We will first, in 3.3.1, discuss what makes a good embedding and setup some requirements for our embedding. We will then, in 3.3.2, describe various approaches to representation learning that we tried and we will discuss why they did or did not work.

3.3.1. What makes a good embedding?

An embedding can have various attributes that effect it uses. For example, how the items in the embedding are distributed could strongly effect algorithms. We will shortly describe some attributes that are important for our application.

The first attribute we will discuss is dimensionality. This describes the number of dimensions that the embedding will have. By making an embedding we are reducing the amount of dimensions. This means that we are losing room for information and we are having to combine that information so that we minimize the information that is lost. On the other hand we require a low dimensional space as to make it comprehensible for humans.

In our system we experimented with two and three dimension, seeing as people are still able to visualize these. During development we tested the effect of using a third dimension. We found that this did allow for better encoding and decoding however the difference was minimal. During development we decided that the clarity and ease of debugging of using two dimension outweighed the slightly improved performance.

The second attribute is the range of the embedding. Seeing as we would like to translate the click-stream sessions onto the embedding as to visualize the user's interests, we require a boundary for the embedding. If we would not implement a boundary we would have to predict the user's interest into eternity. For an embedding the scale of the values for each dimension do not really hold significance, they do not have a unit of measure. For this reason we chose to limit the embedding between 0 and 1 for each dimension.

Distribution is the third and last attribute we will discuss, but it also has a strong effect on the application of the embedding. The main effect that it has on our system is whether the items are distributed evenly or if they are clustered. If the items are spread out evenly it might be hard to determine where the boundaries are between various attributes. Which products are in Spain and which are not? If however the items are strongly clustered, this would mean that for every different value the item should be in a different cluster. Seeing as we only have unique products, this would mean that every product is its own cluster. We therefore want some attributes to be clustered, but some values might also have to be in the same cluster. It seems to be a trade off between evenly distributed and clustered.

A metric that we could use to visualize this is the inter-cluster and intra-cluster distance. The intra-cluster distance, is the distance within a cluster. While the inter-cluster distance is the distance between clusters. While researching we found that there are multiple formulas to calculate both of them. We will use one of the simplest as it does the trick. For the intra-cluster distance we take the average distance from the center of the cluster. For the inter-cluster distance we take the average distance to the global mean. For our embedding it is important that the intra-cluster distance is smaller than the inter-cluster distance.

3.3.2. What did we try?

During development we tried various approaches to see what worked best. We will shortly talk about each of our approaches below.

Using PCA

PCA or Principle Component Analysis is a method that can be used to reduce dimensionality. It finds the lines upon which the items have the most variation, which are called principle components. The first principle component is the line upon which the items vary the most. The second component is the line which has the second most variation, etc. By using the first two principle components we are able to create an embedding that contains the most variation.

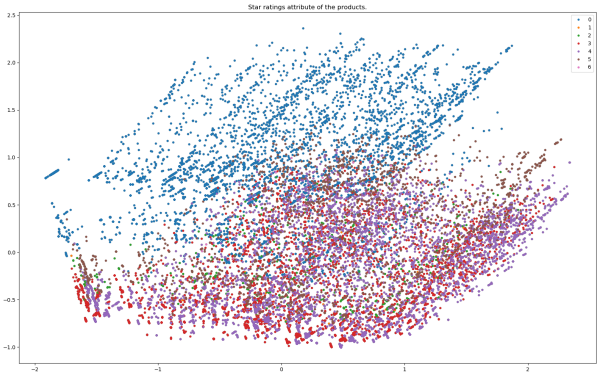


Figure 3.3: An embedding created using PCA. Each product is color coded according to the star rating of the product. We can see that the color groups overlap.

Although it looks like PCA would make an amazing embedding, it actually wont. When using PCA you will often create an embedding that is largely empty. This is because the first and second components often have similar distribution to a normal distribution. This means that a lot of the space in the embedding is for the outliers, while the majority of the items are in one big pile.

Further, seeing as the principle components are lines in the original space which try to encompass the most variance, they do not have clear separation between values. Items that are in Spain will therefor be scattered over a large part of the embedding, together with every other country. In figure 3.3 you can seen an example of this, it shows the amount of star ratings that a product has. As you can see, each group is spread over a large area of the embedding making it rather difficult to distinguish between the groups.

Using autoencoder

An autoencoder is the construction we had in section 3.2.1, where we had an encoder, encoding products into the embedding, and a decoder decoding items from the embedding back to products. If we would decode an encoded product this should result in the same product. Any discrepancies between the product and the decoded-encoded-product must be due to mistakes made by the encoder or decoder model. We could therefore use this discrepancy as feedback for the models in the form of an error. This error could simply be calculated by taking the distance between the product and the decoded-encoded-product. For our models we used Artificial Neural Networks (ANN) as described in section 2.4.1.

During our development we tested a lot with various types of autoencoders, one of the main changes we made to our autoencoders was varying the amount of layers the ANN had. By using more hidden layers we allow the models to process more complex information and make smarter translations (or encodings). If a model has too few layers it is not able to make the necessary conclusions from the data to form a good embedding. On the other hand, the more layers that a model has the harder it is to train the model. The error has to be able to travel through all the layers in order for the model to learn. When it travels through too many layers the error (also known as gradient) starts to vanish, thereby reducing the amount of feedback that the lower layers get. This problem is known as the vanishing gradient problem, which we also ran into in section 2.4.2. One way we tried to mitigate this is by using a stacked autoencoder.

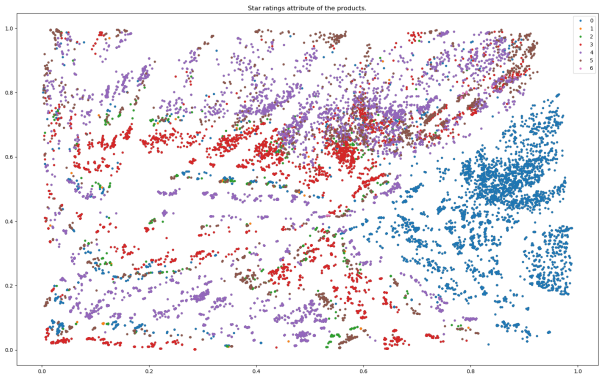


Figure 3.4: An embedding created using an autoencoder. Each product is color coded according to the star rating of the product. We can see that the products form clusters which coincide mostly with the color code.

Stacked autoencoders try to fix the problem of losing the gradient throughout the layers by training one layer at a time. Instead of using the full layer stack from the beginning, it will start by using just one or two layers. This will create an embedding from a higher dimensionality. Once these layers are trained it will add one or two layers and continue training. Instead of training all layers it will now only train the new layers, seeing as the other layers have already been trained. We continue this process until we reach the full layer stack and the dimensionality we wanted for our embedding. By using this process we allow each layer to be trained without having the problem of losing the gradient.

When using the stacked autoencoder, we are able to create quite a promising embedding. In figure 3.4 we can see that the embedding has quite a clear structure which consists out of multiple clusters. We can further see that the clusters are able to divide the attributes of the products, making it easier to distinguish between products based on their attributes. The embedding does have a lot of space between some clusters, especially around the center of the embedding.

Using t-SNE

There was one last technique we tried, T-distributed Stochastic Neighbor Embedding (t-SNE)[28]. Although this is often used as a visualization technique we found that this worked pretty good for our problem as well. T-SNE is a technique that creates an embedding that tries to preserve the relationship

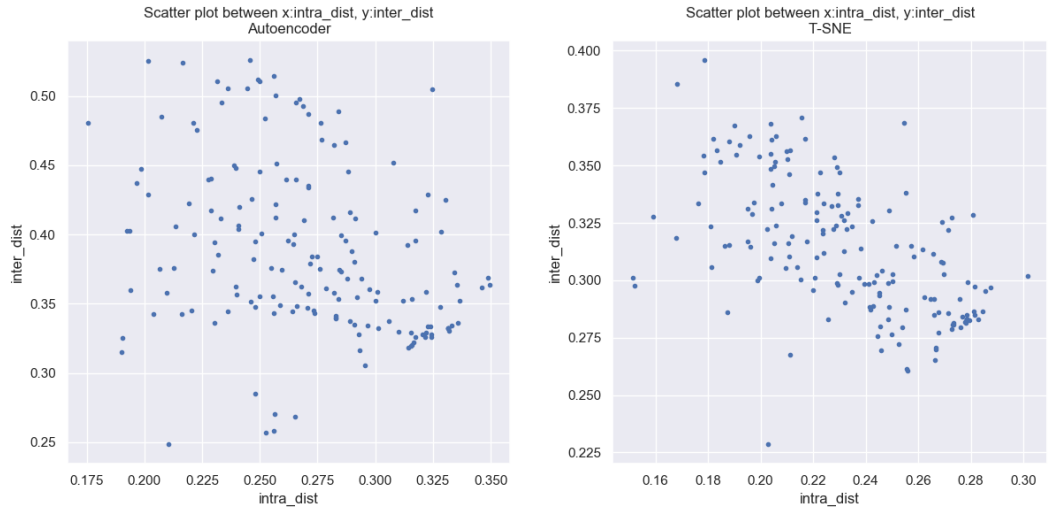


Figure 3.5: Both images display plots between the intra-cluster and inter-cluster distances. The left image is for the autoencoder, the right image for t-SNE.

between points. It accomplishes this using two steps. Step one is to create a probability distribution that encompasses the relationships between the points in the original space. Then, in step two, we try to create an embedding that adheres to the probability distribution.

By applying this technique we get an embedding that seems quite strongly clustered and with a significant amount of emptiness along the edges, as depicted in figure 3.6. To tell if this is a good embedding we needed to look at the metrics that we described in section 3.3.1, inter- and intra-cluster distance. Figure 3.5 shows the intra- vs inter-cluster distances for the autoencoder and t-SNE. We can see that both distances are smaller on for t-SNE and we can see that both models have all there points to the left of the x=y line, which is good. The lower inter-cluster distance for t-SNE shows us that the clusters are further together probably centered around the middle. The lower intra-cluster value means that the clusters are smaller.

Looking at figure 3.6 we can see the embeddings that are learned using both of the approaches. We can clearly see that the embedding that was learned using t-SNE has more empty space along the edges, while the one created using the autoencoder has more in the middle. Although the metrics do not really state a clear champion, the scatter plots do. The embedding created using t-SNE has way more structure in the embedding then the embedding created using the autoencoder. This structure means that the products are grouped more strongly while they also still have products in between. This is further confirmed by figures A.1 and A.2. We see that the first figure, displaying the t-SNE embedding, has tighter boundaries between the two clusters than the embedding created by the autoencoder. Although we see some images that do not divide the embedding at all. We can still stat that T-SNE seems to be better for the task at hand, even though it leaves the corners of the embedding empty.

3.4. Understanding user interests

To be able to make a recommendation it is crucial that we understand what to recommend. The heart of this problem is understanding what the user wants. We can split this question into several smaller questions: What user interests are and how they can be represented, which we will discuss in section 3.4.1. We will then continue in section 3.4.2 by describing how we can learn these interests using models. Finally, in sections 3.4.3, we will discuss the solution that we implemented and why we chose to go for that solution.

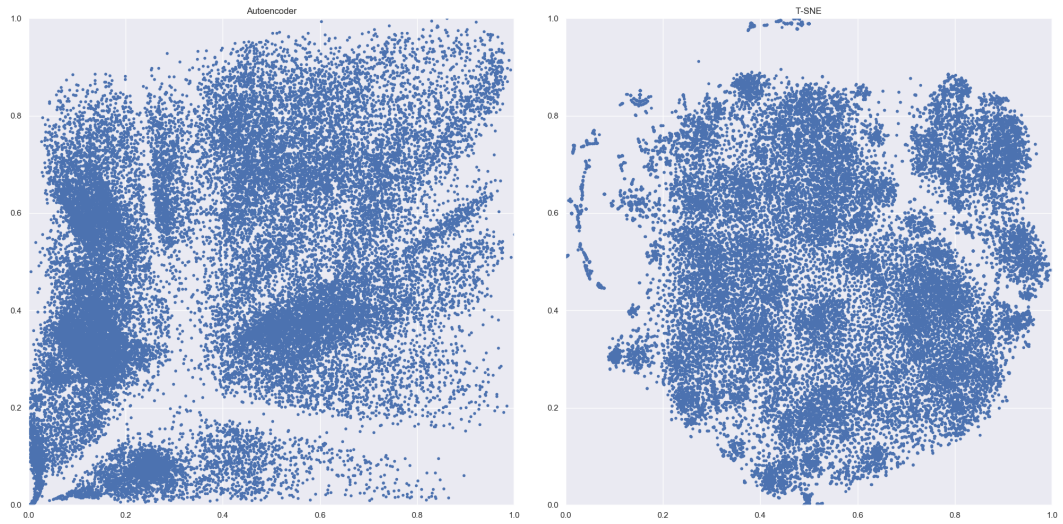


Figure 3.6: Both images display scatter plots of the products in an embedding. The embedding in the left image is created using an autoencoder, the one on the right using t-SNE.

3.4.1. What are user interests and how can we represent them?

If we want to make good recommendations, it is key that we understand what the user is interested in. When we are looking at e-commerce the user's interests are the products he might want to buy. This is in our case also our potential sale. If however, the user does not find the product he is interested in, we will not make our sale. It is therefore key that we understand the user's interests and provide them as fast and easily as possible. A user's interest could also be more general than a specific product, instead of a certain hotel it could also be hotels in Spain, or child friendly hotels.

To understand what the user's interests are we have a couple of options:

- Ask the user.
- Make an (educated) guess.
- Deduce what they are.

Seeing as the user has the option to search, this option is mostly covered. We do of course still have the interests that the user is not aware of and the ones that are hard to find. To make an educated guess is a fair option if we do not have any better options. Presuming that the power-law is in effect, only a small part of our products are most likely responsible for the majority of our sales. Recommending one of these products could be a good start.

The final and perhaps most difficult option is deducing the interests. To do this we require some data about the user. In our case we can use the clickstream data for this. The idea being that we analyze the user's behavior on the website and somehow deduce which interest he most likely has. We can then look which products best fit his interests and which products we should therefore recommend. This plan still misses a few key parts:

- How are we going to analyze the user's behaviour.
- How are we going to represent a user's interest.
- How are we going to deduce the interest from the behaviour.

We do not have a lot of options for analyzing the user's behaviour, given that we only have two datasets to our disposal. The first dataset has product information, this wont help us a lot with analyzing user behavior. The second dataset is clickstream data, which might just be perfect for this. Representing user's interests isn't that straightforward though. Seeing that a user can have an interest

that spans multiple items, such as 'hotels in Spain', we need a way to encompass all the products.

Each product needs to have a value indicating the interest the user has in it, a sort probability distribution. To realize this, we can generate a heatmap that encompasses the embedding. The heat that a certain item in the embedding has, corresponds to the interest that the user has for the product. For each user we will then generate a heatmap, which we can analyze to get the products that we should recommend.

The heatmap would consist out of some sort of pixels. Given the resolution of the heatmap, the pixels could either contain multiple products in a single pixel, or some pixels could not have any. The higher the resolution, the smaller the pixels, and the harder it is to train. A higher resolution means each pixel has less feedback seeing as it contains less products. The goal therefore is to have the resolution be as high as possible while still being trainable.

The final component is how we deduce the interest from the behaviour. In other words, how do we generate a heatmap for the embedding from the clickstream data. We will discuss the various options that we tried in the following section, section 3.4.2.

3.4.2. How can we learn user interests?

To learn the user's interests and generate the heatmap, we need to know two main things. What model should we use and how can we train the model. We will start by answering the second part, how we can train the model. Once we understand all the information that we will feed into the model, we will talk about the model we are going to use.

Training the model

To train a model, we need to be able to provide it with some sort of feedback, some sort of an error. Seeing as we do not know the user's interest, we are going to have to deduce them in order to train the models.

From the clickstream data we know which products the user interacted with, we also know which products the user eventually purchased. These are both clear indicators that the user was interested in these products, therefore these products should have a lot of heat in the heatmap. Of course the level of interaction says something about how interested the user was to the product. A simple click on the image is clearly less interest than filling in personal information. This in turn is less interest than an actual purchase. We can therefore award different levels of heat for different levels of interaction. Using this method we can then create a heatmap that represents at least a part of the user's interest for the sessions that include a purchase.

The difference between the predicted heatmap and the reconstructed one can function as error for the models. This allows us to train the models to give a lot of heat to the products that will probably be purchased and less to those that probably wont be purchased.

Using recurrent neural networks

One of the most obvious models to experiment with was the Recurrent Neural Network RNN (2.4.2). Because the clickstream data is comprised of a sequence of events, it makes sense to use a sequential model for this task. The RNN gave us some promising initial results. When further exploring these models, we found that the LSTM and GRU models trained a lot easier than the RNN model. They were able to get better results in a shorter period of time. The GRU model was able to train the fastest, while maintaining the same quality results. Especially the optimized 'CuDNNGRU' version of the GRU model was very fast.

When testing the network we found that the initial results of the RNN seemed promising. However when we further increased the dimensions of the heatmap, we found that the model had a strong tendency to 'collapse'. With this we mean that it would leave the heatmap empty and just fill it with zeroes, because this would yield a lower error. This is because the true heatmap that we calculate

becomes sparsely filled when we increase the dimensions. For example, if a session would only interact with four hotels, then this would be fine with a dimension of 5 by 5 seeing as this would result in approximately 4/25 pixels being filled. When we increase the dimensions to 10 by 10 or more this becomes a problem seeing as only a few percent of the pixels would be filled. The model therefore takes the easy way out and predicts only zeroes and only has the error for misclassifying those few pixels.

We try to solve this problem in two ways. First we try to give heat to more pixels in the true heatmap. We do this by not only using bookings as labels but interactions with hotels of any sort. When, for example, a user views a page this is seen as an interaction. We do of course give less weight to small interactions compared to bigger interactions such as booking the hotel. The second improvement that we made is to give weights to the misclassifications. When the model misclassifies a zero pixel in the true heatmap by giving it some heat, this is not a problem and the error is just the same as before. If however, the model misclassifies a pixel with heat in the true heatmap by giving it less heat, then the error will be the normal error multiplied by the specified weight. By increasing the weight we can ensure that the model will not disregard those few filled pixels in the true heatmap. By implementing these two solution, we were able to increase the resolution to 40 by 40, which is significantly higher than before.

Using convolutional neural networks

Instead of sequentially going over the data, we wanted to explore the possibilities of looking at all the data at once. The problem with this is that the sessions do not have a fixed length. This meant that the input length for the model would be different for every session. We mitigated this problem by choosing a fixed size and cutting the rest of the data off. If the session was shorter than the input size, we would pad the data with zeroes. This allowed us to experiment with non sequential models.

We wanted to try using a convolutional network (CNN), as described in section 2.4.3, due to the increased speed of training. This would allow us to train the models faster allowing us to process more data in a reasonable amount of time. We used one dimensional convolutions on the two dimension data in order to allow us to process the data in a single dimension. Because of the multiple kernels, we still ended up with two dimensional data after every one dimensional convolution. After several convolutions we use a global pooling layer to combine the results into a one dimensional feature. We then proceed by applying a traditional ANN (2.4.1), after which we reshape the feature to form the heatmap.

We also experimented with various methods that would allow us to increase the capabilities of the CNN, one of which is the residual network[29]. A residual network contains residual connections that allow values to skip layers. This has the benefit of allowing the gradient to flow more easily and therefore making it easier to train. Because of the many layers it is still able to process complex tasks, while also allowing it to be trained easier.

Eventually we concluded that we were not able to understand the user's interests not due to the models, but due to the amount of data in the embedding. The embedding contained so much information that a lot of information loss was occurring. We therefore decided to remove some information from the embedding to hopefully alleviate the problem.

3.4.3. What did eventually work?

In order to reduce the information encoded into the embedding, we decided to remove the country features and to include them separately into the models. We further decided to clean the dataset more before hand, to thereby also reduce the information that needed to be encoded into the embedding.

We included the country data by splitting the model, described in section 3.4.2, into two separate models. This means that we now had two models that when put together should predict the user's interests, as shown in figure 3.7. The two new models are:

- **Heatmap model:** This model is quite similar to the model we introduced in section 3.4.2. It uses the clickstream data as input and uses the embedding as feedback. It in constructed using both CNNs and ANNs and is depicted in the left image of figure A.3.

- **Country model:** The second model also uses the clickstream data as input, however it does not use the embedding as feedback. Instead it uses the countries of the products that the user interacted with as feedback to train. This model is also based on CNNs and ANNs and is depicted in the right image of figure A.3.

The output from the two models (the heatmap and the country prediction) need to be combined afterwards, in order to make recommendations. Both the heatmap model and the country model, have a similar layout. This is because they need to process the same data, the clickstream data. They both initially use one dimensional layers, after which they group the data using a global pooling layer. Finally they run the results through two ANNs to yield their prediction. The structure of the two models is shown in figure A.3.

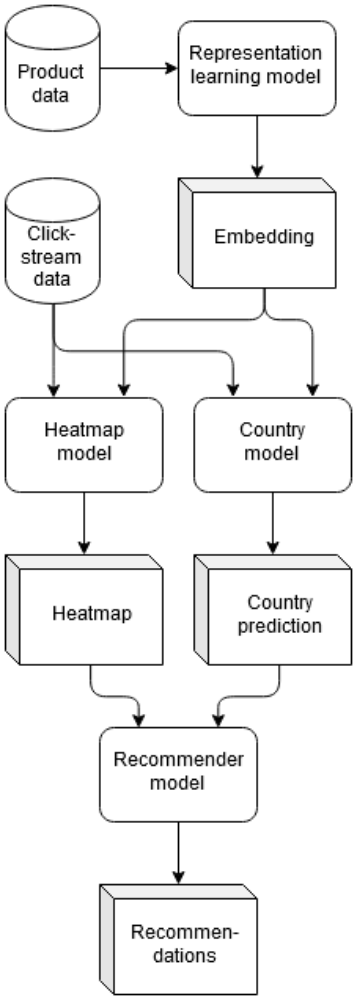


Figure 3.7: Depicts the new system including the split heatmap and country model. The cylinders depict the data sources, the boxes represent results from subsystems, and the rounded rectangles depict the models.

3.5. Making recommendations

Now that we have the heatmap and the country prediction visualizing the user's interests, we only still need to make the actual recommendations. In order for us to do this properly, we must first understand what we want to achieve with the recommendation. We will discuss this in section 3.5.1. After this we will discuss our implementation in section 3.5.2.

3.5.1. What do we want to recommend

Now that we understand the interests of the user it seems trivial to make the recommendations. We simply take the products with the most heat and recommend them. The problem with this is that these products will be very similar, the user might want to have some variety in the recommendations. We might have multiple hot spots, by using this approach the other hot spot s will not be recommended. We decided to try and split our priorities, on the one side we would try to exploit the user's interests by recommending products from the hottest spots in the heatmap. On the other side, we would try to explore the heatmap, recommending products from various spots on the heatmap. We are therefore faced with a trade-off, exploration vs exploitation

Another problem arises when we consider that we also have the country prediction. How are we going to combine them? We want to recommend products with both a highly predicted country and a lot of heat. Because of this we decided to combine them, we add them together and use a weight to define the proportions of the two. By changing the weight we can make the product's heat depend more on the country prediction or on the heatmap.

3.5.2. Making the recommendations

For making the recommendations we decided on a sequential process. This would allow us to make as many recommendations as we wanted, it also seemed like an intuitive approach seeing as the first recommendation is fairly trivial.

The first recommendation is the product with the most heat (defined by both the heatmap and the country prediction). Recommending the second problem starts to get interesting. We do not want to recommend the first product again, therefore we set it's weight to zero. We also know that we do not want to recommend a neighboring product because we would like to explore a bit to ensure some

variety. However we also do not want to put these weights to zero, because this might just be one of only a few high spots so we might need to come back to recommend some of them. We therefore just reduce the heat depending on the distance from the first recommendation on the heatmap.

To determine the heat reduction we calculate the distance and then scale it to ensure that all the distances are between one and zero. We then flip the value by subtracting it from 1 and insert it into the Gaussian formula. This creates a nice dent/crater surrounding the first recommendation, which we apply to the heat of the products using multiplication. We are able to change the size of the dent by changing the scaling factor. we can change the dept by changing the value for sigma in the Gaussian formula.

Now that all the heat has been adjusted we can simply make a second recommendation by taking the product that now has the most heat. We can then repeat the steps but now with regard to the second recommendation instead of the first. By repeating this we will both exploit and explore the heat of the products.

4

Evaluation

In order to understand the quality of our results, it is important to evaluate them. By doing this, we will get a better indication of whether the system performs as expected. We have three components of which we should evaluate the results. These being the embedding, the representation of the user's interests, and finally the recommendation themselves. It is important to understand that if one of these components has bad results, then this will translate to the latter components that use these bad results. We will discuss the components in sections 4.1, 4.2, and 4.3 respectively.

4.1. Embedding

In order to be able to evaluate the results of the representation learning system, we must first understand what the desired results are. In section 3.3.1 we described some of the attributes that we expect from an embedding, the most important one for evaluation being the distribution of the products in the embedding. We also introduced a metric that could visualize this, the inter-cluster and intra-cluster distances.

We will first discuss the use of visual characteristics and cluster metrics for evaluating the embedding in section 4.1.1. After which, in section 4.1.2, we will introduce a new distance metric to further evaluate the embedding using the clickstream data.

4.1.1. Visual characteristics and cluster metrics

In section 3.3.2, we compared various implementations of the representation learning system. We evaluated them using the visual characteristics of the embedding and using the inter-cluster and intra-cluster distances to show how the products were distributed. Using the visual characteristics we could rule out methods such as PCA. We could clearly see that the product attribute categories overlapped, making it hard for the system to distinguish between them.

When the visual characteristics were not sufficient, as can be seen in figures A.1 and A.2, we used inter-cluster and intra-cluster distances to compare the results. In figure 3.5 we can see the scatter plots depicting the values for each of the product's attributes. By comparing these we can see that they are quite comparable. The only big difference is that both distances are smaller for the t-SNE network. With that finding, in combination with the visual inspection, we found that t-SNE outperformed the autoencoder based model.

4.1.2. Distance metric

To truly evaluate the representation learning system, we measured how nearby products from a user's session were compared to a random sample. For this we selected 200 random sessions from the clickstream data. For each session we collected all the distinct products that the user visited. We then continued to sample the same number of random products in order to form our comparison. We compared them using our t-SNE learned embedding and two distance metrics.

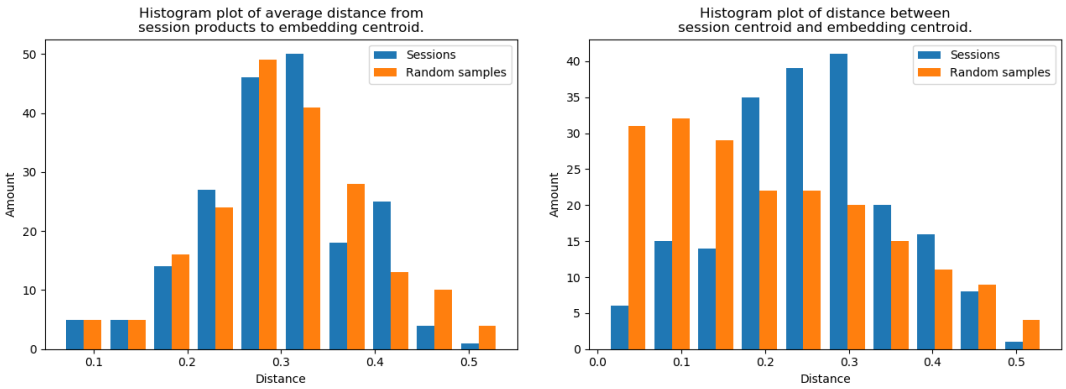


Figure 4.1: Compares the distance between products from a session and products from a random sample. Compared based on distance to the center of the embedding. Left image uses the average of the L2 distance for each of the products. Right image uses the L2 distance between the center of the products and the embedding center.

The first distance metric is the average euclidean distance between each product in the list and the embedding's center (average coordinate). In the left image of figure 4.1 we can see that the results are quite comparable; they both seem to follow the same distribution. This is because they most likely follow the same distribution, seeing as the products originate from the same embedding.

The second distance metric uses the average coordinate (centroid) of the products in the list and compares this with the centroid of the embedding. With a random sample these distances should be small, because the individual products cancel out each other, giving us a centroid close to the embedding's centroid. If the learned distribution of the products in the embedding successfully grouped similar products together, then we should see bigger distances for the products from the sessions. This does presume that the products in a session are somewhat similar.

When looking at the right image of figure 4.1, we indeed see that the products from the sessions have larger distances than the randomly sampled products. This shows that the representation learning system was successfully able to learn an embedding that places similar products closer together.

4.2. User interest

The second model in our system tries to represent the user's interests. To evaluate its quality, we must first understand what a good representation entails. In section 3.4.1, we described what user interests are and how we can represent them. We continue in section 3.4.2 by describing how we can give feedback to the models in order to improve the results. From this we can conclude that we want a heatmap that has high values for the products that the user eventually purchases and lower values for the remaining values. We similarly conclude that the country prediction should have high values for the countries that correspond to the products that get purchased, and low values for the other countries.

4.2.1. Loss function

One of the ways in which we evaluate the quality of the heatmap and the country prediction is during the training of the models. As depicted in figure 4.2 the model will stop every few steps to evaluate its progress. During this evaluation step we provide the model with unseen clickstream data from the evaluation dataset. This ensures that the results will be comparable to when it is in production with live clickstream data. We then evaluate the results using the same loss function as used for training.

The loss function, as explained in section 3.4.2, compares the results to a custom generated heatmap that only has heat for the pixels with the products that are interacted with during the session. For the country prediction the loss function simply is the L1 distance between the prediction and a custom vector that has zeros except for the actual countries which were viewed during the session.

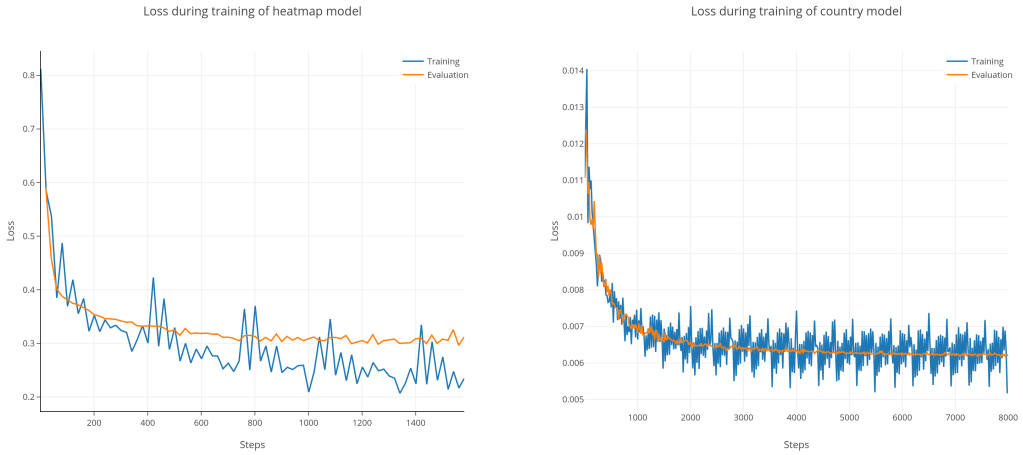


Figure 4.2: The loss during training, both on the training dataset as well as on the evaluation dataset. The left image depicts the training of the heatmap model while the right image depicts the country model.

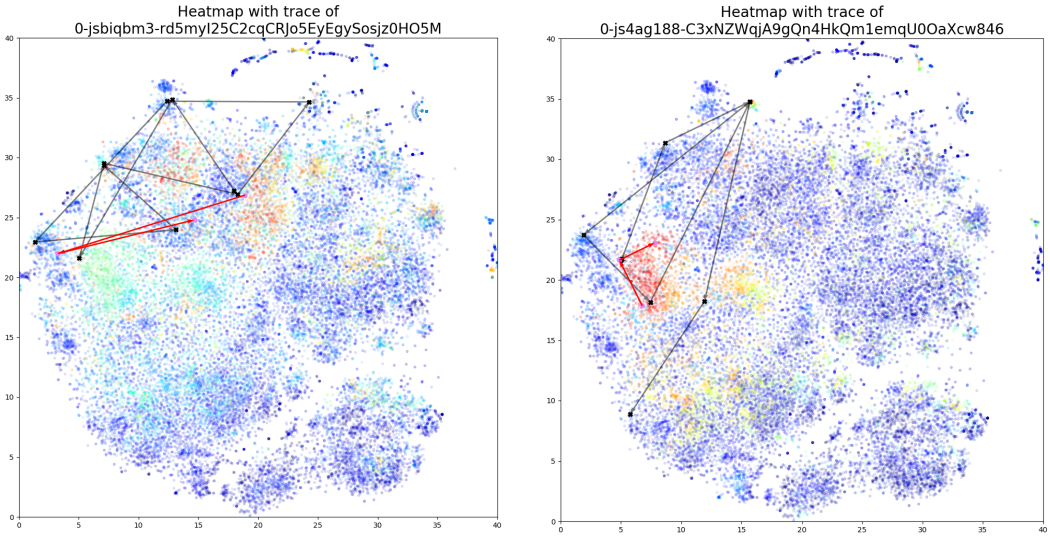


Figure 4.3: These scatter plots depict the products, with the color representing the heat. The red arrows depict the users interaction in chronological order, and the black arrows represent the recommendations.

In figure 4.2, we see that the heatmap model starts to overfit after only a few hundred steps. The country model, however, seems to be able to run for thousands of steps, slowly improving the results without overfitting. For us this meant that we were only able to train the heatmap model until a certain degree, while the country model could be trained for a long time. Both models seem to be learning, which should be a good indication regarding the results.

4.2.2. Visual

Using a heatmap allow us to visualize the user's interests. This also allows us to evaluate the heatmap by visually inspecting the results. In figure 4.3 we depict the products as they exist in the embedding. We further color them according to the heatmap and country predictions as described in 3.5.1. Finally, the images also show the products that the user interacted with in red, and the recommendations made in the next step in black.

Using this visualization we can clearly see that the products the user interacted with are positioned close to each other and in a region with quite a lot of heat. We can also see areas in the heatmap that

might contain products of interest that the user did not visit. Not only does this allow us to evaluate the heatmap and country models, but it also allows us to understand potential improvements to the system. Looking at the two images, it seems that the models are generating promising results.

4.2.3. Session heat

Our final metric will use the unseen clickstream data to calculate the average heat for a session which it can then compare to a random list of products. We first randomly selected 500 sessions from the unseen clickstream data. This means that the model has no way of knowing that it should further increase these regions, except for receiving the first part of the session. We then use the heatmap model and the country model to predict the corresponding heatmap and country prediction. Which we continue to use to generate heat for each product as described in section 3.5.1.

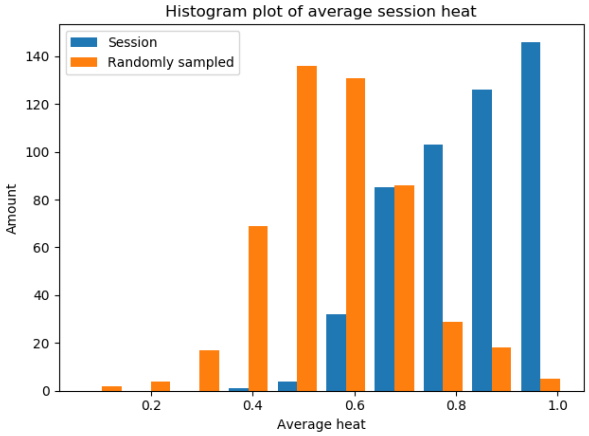


Figure 4.4: The average heat from the heatmap combined with the country prediction, for a session products and a random sample.

To evaluate the system we take the products that occur in the session, and we take a random sample of products of the same length. We then calculate the average heat for both of these products list, which results in distribution as shown in figure 4.4. We can clearly see that the random samples form a sort of normal distribution, while the sessions linearly increase until they reach a heat of 1. We can therefore state that although the distributions overlap, the average heat of a session is definitely higher than a random sampled list of products. This indicated that the heatmap and country models are able to capture the user's interest to a certain extent.

4.3. Recommendations

The evaluation of the recommendations is also the evaluation of our system and of the effectiveness of the embedding for recommendation. For the evaluation it was important that we only use the recommendations and the data available to the models; we could not use any intermediate results. We set up two methods to evaluate our system. The first method is based on metrics, while the second asked users to rate the systems. We will discuss the systems in sections 4.3.2 and 4.3.3. However, first we would like to shortly explain our setup in section 4.3.1.

4.3.1. Setup

In order to properly evaluate the system, we compared it to the current recommendation system used by Vakantie.nl. To ensure that the comparison between the two systems was fair, we had to make sure they had similar environments. We started off by selecting the clickstream data that was available for each of the current system's recommendations, and only providing that section to the user interest model. The recommender system then recommended the same number of hotels as the current system recommended to ensure a fair comparison. We further ensured that the evaluation methods only used the IDs of the hotels. The rest of the information was gathered from the product- and clickstream datasets.

4.3.2. Metrics

The first method to evaluate the results of the recommender system, is to extract metrics from the recommendation with which we can compare the system. For this comparison we looked at 100 recommendations and only used the attributes of the hotel to compare.

To compare the country prediction, we simply looked whether the country of the recommended

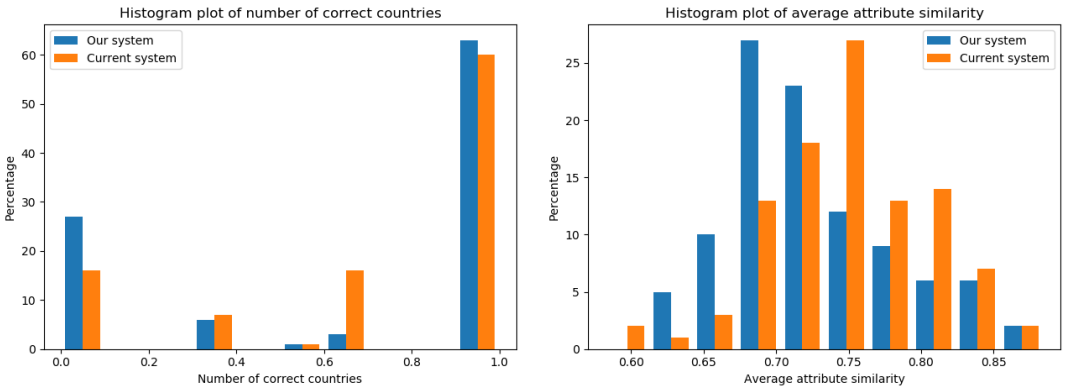


Figure 4.5: In the left figure we show a histogram of the percentage of correct recommended hotels in regard to the country. In the right image we show a histogram of the average similarity between the recommended hotel and the actual booked hotel.

hotel was in the list of countries of the hotels with which the user had had an interaction. If the user interacted with a hotel in the country of the recommended hotel, we gave a one, otherwise a zero. We then took the average of the scores per recommendation. These recommendations mostly consisted out of three recommended hotels. However, this was not always the case. Once we calculated the average for each of the recommendations, we could then depict the results using a histogram as shown in the left image of figure 4.5. We found that the current system has an average of 0.78 while our system only had an average of 0.72. Although this might not be a big system, this still shows that our system does not have a good enough grasp on recommending the country the user is interested in.

The second metric that we used to compare the recommendations, compared the attributes of the hotels together. We used $1 - L1\ distance$ as a similarity metric in order to visualize how the systems compared, with regard to the other attributes of the hotel. We then took the average of all the hotels of a recommendation. When we plot these results we get the right image of figure 4.5. We can see that once more the current system performs slightly better. When we look at the average value we see the same thing. The current system had an average similarity of 0.75, while our system has 0.72. Although these results are not promising they are still rather close, we therefore decided to set up another evaluation.

4.3.3. Preliminary user study

For our second evaluation method we provided 12 potential users with the recommendations. For this we used 15 sessions from the historical data, including a recommendation from the current system. We then created our own recommendation as described in 4.3.1. We provided the participants with the clickstream up until the recommendation. The participants therefore only had the events available prior to the current system's recommendation. As depicted in figure 4.6, the participant were provided with the two recommendations, randomly named 'Recommendation A' and 'Recommendation B'. The participants were asked to choose which recommendation was better, given the session with which they were presented. Their choices were: 'A for sure', 'I think A', 'I do not know', 'I think B', and 'B for sure'.

The participants were between 20 and 70 years of age with the majority being between 20 and 30. Only a quarter of the participants were female, and almost all of the participant have a university diploma or are currently studying at a university. Although this might not be the normal demographic for a user of Vakanties.nl, this should still give an indication of the behaviour of a typical user.

To get to an evaluation, we unscrambled the recommendations so that it was clear which system they voted for. We then scored the choices. Each 'I think' was worth one vote and each 'for sure' worth two votes. This gave us a number of votes for system A and system B for each of the session as shown in table 4.1. We further calculated the portion of votes for our system, which resulted in an average of 0.26. These results indicate that the current system of Vakanties.nl was able to generate

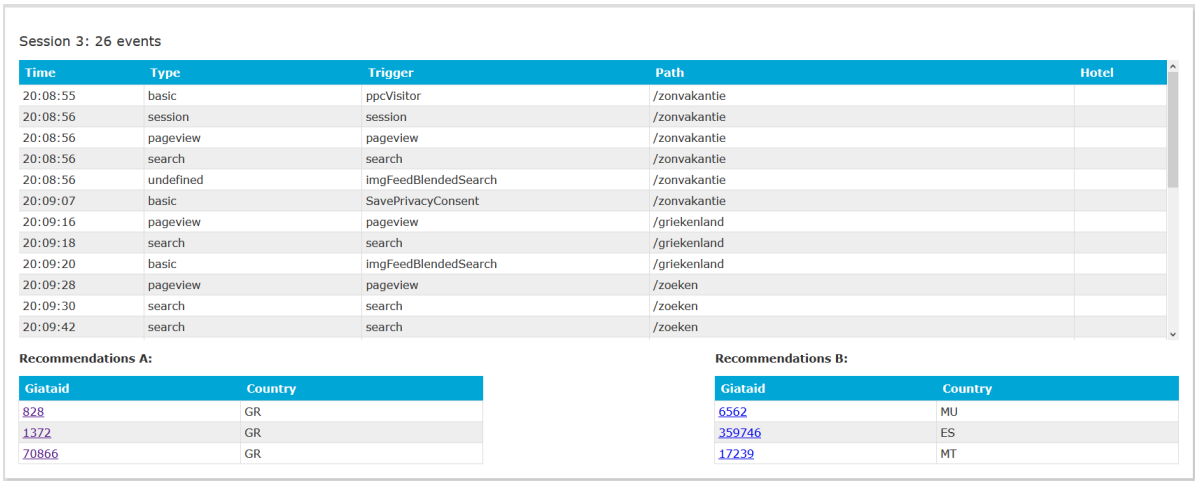


Figure 4.6: The interface that the participants were presented with.

Table 4.1: Results of the preliminary user study. Votes of the participants for each session

	Our system	Current system	Portion
Session 1	3	13	0,188
Session 2	1	11	0,083
Session 3	21	0	1
Session 4	5	6	0,455
Session 5	19	0	1
Session 6	2	13	0,133
Session 7	16	2	0,889
Session 8	1	12	0,077
Session 9	1	8	0,111
Session 10	1	15	0,063
Session 11	1	17	0,056
Session 12	0	15	0
Session 13	0	16	0
Session 14	0	12	0
Session 15	0	19	0
Total	71	159	0,270

recommendations that appealed more to the participants than our recommendations. We should keep in mind that the participants might not be a representative sample of the users.

The participants commented that the sessions often had a clear country on which the user was focused, while often one of the recommendations only had hotels in another country or region. Looking at the session, we see that the sessions in which all users voted for the same system often correspond with these findings. Although this could be due to a bad system design, the more likely cause is that it is because of a lack of regional features as input. During development we made the decision to remove some of the features, as mentioned in section 3.4.3. During this process the regional features got excluded from the embedding but did not get added to the country encoding. This was because it would be difficult to encode the regional features into the encoding in an efficient manner. Therefore, the system did not know the region that the customer was interested in, which would explain the results of the preliminary user study. We will discuss this further in section 5.2.1.

5

Conclusion

In this chapter we will conclude our report. First, in section 5.1, we will look at the research questions and evaluate whether we have answered them to fruition. After this, in section 5.2, we would like to suggest some improvements that can be made to the system to improve the results. We will start off by summarizing the thesis.

5.1. Evaluation of the research questions

In section 1.3 we introduced the research questions. These consisted out of three main research question and various sub-questions. We will evaluate the three main research questions in sections 5.1.1,5.1.2, and 5.1.3 respectively.

5.1.1. How can we create an effective embedding in order to encode the products of Vakanties.nl?

Before we can evaluate whether we answered the main question, we will first have to look at the two sub-questions.

Which criteria are important for an embedding in order to encode the products of Vakanties.nl?

In section 3.3.1 we described a couple of attributes for our embedding, namely: dimensionality, range, and distribution. By ensuring that these attributes are present in the embedding, we enable it to be used not only to encode the products but also to represent the user's interests.

How can we encode and decode products to and from the embedding?

As described in section 3.3.2, there are several methods that could be used to encode products into the embedding. In this section we continued by comparing the methods and found that t-SNE was the best method for our criteria. Although t-SNE does not allow products to be decoded from the embedding, we still found this to be the best solution. By redesigning our system, as described in 3.2.4, we found that we no longer had a need for decoding products from the embedding. We did introduce another method that is able to both encode and decode products from the embedding, namely the autoencoder.

Main question

We showed that we answered both sub-questions by setting up criteria for our embedding and introducing methods that could create an embedding that satisfied these criteria. By answering these questions and implementing the method in question, we were able to create an embedding that met the criteria. In section 3.3.2 and section 4.1 we further discussed the quality of the embedding and found that it was effective at representing the products.

5.1.2. How can we validate the effectiveness of the embedding for making recommendations?

We will again first look at the two sub-questions before evaluating the main question.

How can we use the embedding in combination with clickstream data to learn which products the user is interested in?

We introduced two methods that would use both the clickstream data and the embedding in order to predict which products the user was interested in. In section 3.2, we introduced the first method. This method would both create the embedding and use it to recommend a region of the embedding that the user was interested in. As described in section 3.2.3, we found a couple of problems with this method. The most important one being that it could only recommend a single product.

We continued in section 3.4.3 by introducing the second method which consisted out of multiple components. One of which uses the clickstream data in combination with the embedding to represent the user's interests. This method is described in 3.4 and predicts the user's interest in the form of two predictions: a heatmap over the embedding, and probability for each country. We evaluated the results of these two models in section 4.2, by combining the results we seem to be able to get a good indication of the user's interests.

How can we use the interests of the users to make recommendations?

The second sub-question refers to the need for a second method that translates the user's interests into an actual recommendation. This method is introduced and discussed in section 3.5. We start off by first addressing the difficulty of the task and introducing some trade-offs that need to be considered. After which, in section 3.5.2, we propose an implementation that combines the two predictions mentioned in the previous sub-question to generate a heat for each product. It then uses this heat to continuously generate recommendations. This approach has the added benefit of being able to generate as many recommendations as needed. The use of heat further allows it to be easily visualized using the embedding.

Main question

As mentioned in the two subsections, we created a system to make recommendations from an embedding using clickstream data. Seeing as this system depends on the embedding, therefore the results of the system also reflect back on the effectiveness of the embedding. By evaluating recommendations and the two subsystems corresponding to the two subquestions. We can get an idea of the effectiveness of the embedding for making recommendations.

In chapter 4 we evaluated the three systems. We found that the results of two subsystems were promising, indicating that they are able to perform their tasks. The third system, which is responsible for the actual recommendations, had results that were less promising. When comparing it to the current recommendation system of Vakanties.nl, we found that their recommendations were slightly superior to our recommendations. This can be due to multiple reasons, a couple of which we mentioned in section 4.3.3 which we will further discuss in section 5.2.

5.1.3. How do the recommendations made while using the clickstream data and the embedding compare to the recommendations made by the model currently used by Vakanties.nl?

In section 4.3, we saw that we evaluated our recommendations using two methods, metrics and with a preliminary user studies. The metrics showed that although the results were quite similar, the current system seemed to have the edge. This in part can be due to a lack of data provided to the model, or a lack in the system itself. When looking at the findings of the preliminary user study, we see that again the current recommendations is voted to be the preferred system. We further find that this is largely due to a lack of regional data for our system.

We should however note that although the evaluation was less promising than we hoped, the system we developed still has a few advantages compared to the current system in use by Vakanties.nl. First off, by having their own recommendation system they no longer have to be dependent on another company for their recommendations. This further allows for more flexibility, as they can change the system as they wish. The second advantage that we would like to mention is that our system allows for certain insights as to why a product is recommended. By using an embedding we can get a clear idea as to the relationship between products. This is not only useful for understanding why a recommendation

is made, it can also be used for various other tasks. As an example, Vakanties.nl could use it to understand which products it should offer or to personalize its advertisements.

5.2. Future work

As a final note we would like to discuss some improvements that could help improve the quality of the system. These are improvements for which we did not have enough time to implement them or improvement which we only noticed after the evaluation of the system. We grouped the improvements in three categories: improvements to the data, improvements to the models, and improvements to the evaluation. We will discuss them in the sections 5.2.1, 5.2.2, and 5.2.3 respectively.

5.2.1. Improving the data available to the models

In order to make good recommendations we need models that are well trained. To get well trained models we need good data. Therefore a good working system all depends on good data. The first two improvements are therefore regarding the data that we used to train the models. We will go over each separately.

Regional features

The first potential improvement as mentioned in section 4.3.3, is to add the region and subregion features back into the datasets, perhaps even add the city. During development we made the decision to exclude the region and subregion features from the datasets that we use to train and evaluate the models. We decided to do this because it was rather difficult to encode these features in such a way that they did not take up too much space in the encoding. The lack of region and subregion features means that the models do not know in which region a hotel resides. This is a problem because regions can be quite important during the recommendation of hotels. A hotel in the south of Spain on the coast is quite different then a hotel in the central or northern regions of Spain. We also saw this during our experiment. Although the system understood in which country the user was interested, it did not understand the region. This is because the models did not have access to the information.

To mitigate this problem we can either incorporate these features back into the hotel attributes. Or we can expand the country encoding to also include the region. It would seem that it would make more sense to include the regional feature into the country encoding seeing as these features both describe where the hotel resides. The regional features can even replace the country feature to a certain extant. Once the define the region, the country is also specified. This might not be worth while for small countries or countries that have very few hotels.

Product popularity

Given our two datasets we used almost all the information to our disposal, one of the features we could still use is the popularity of a product. Given the historical data we could calculate the popularity of each of the products; how often it has been viewed and how often it has been booked. We could incorporate this information in the recommender component. By knowing the popularity of items we can choose to boost their heat a bit seeing as the have a higher probability of being booked given the historical data.

Another application might be to normalize some of the data for the other models. The heatmap model will for example see the popular products more often than the non popular models. The model therefore has a bias towards the popular models. By normalizing the data before hand we can have control over this bias. We are then able to control the weight of the bias.

5.2.2. Improving the models and their results

After we ensure that are data is in good condition, we must ensure that the models are able to make the predictions that we want. To improve the results we have a couple of potential improvements to the models. We will go over the improvements in the same order as they occur in the system.

Embedding

The first model that we run into is the representation learning model. Although the t-SNE model works rather well there are still some adaptation that we should try. The first adaptation which we also

mentioned in section 3.3.1, is to increase the dimensionality from two to three. During development we tested a three dimensional embedding but did not see big improvements and therefore opted for the two dimensional embedding. Since then we have changed a lot of parts of the system. We therefore think that this is worth trying and could improve the amount of data that is stored in the embedding.

The second adaptation in regards to the embedding concerns the distribution of the products. As discussed in section 3.3.2, using t-SNE creates an embedding that has a clustered structure with a lot of emptiness along the edges. This means that when we use the embedding to create a heatmap some of the pixels will be useless. To mitigate this we should transform the embedding to 'push' the products further into the corners. This should lower the products per pixel for the pixels in the middle while increasing for the pixels along the edges.

User interest

After the representation learning step we get to the model responsible for understanding the user's interests. This model generates the heatmap and the country prediction. One of the ways in which we could improve the quality of the heatmap is by increasing its resolution. By increasing the resolution we decrease the number of products per pixel. This allows the model to represent the user's interest with higher accuracy. The difficulty with this is that it is also harder for the model to understand which pixels need to get heat. To accomplish this we would need to train the model using more data to ensure that it does not overfit and learns the user's interest.

Recommender

Finally we get to the last model, the recommender. As described in section 3.5.1, one of the difficulties with making recommendations is the trade-off between exploitation and exploration. On the one hand we want to exploit the products we know he is interested in, while on the other hand we should also introduce the user to products that he might also like. With our current implementation, we have a lot of focus on exploration, while we might need to focus more on the exploitation half of the trade-off.

5.2.3. Improving the evaluation process

Finally after improving the system itself it is important to evaluate whether the changes have actually led to better results. For this we can use our current methods of evaluation, however as stated in section 2.2.3, the only real way to test a recommendation is to actually put it in production. Although it might be beneficial to first validate the results on historical data, the only way to get a real understanding of the quality of the results is to apply A/B testing.

A

Additional data

In this section we depict some of the larger images and tables, which are only referenced occasionally.

Table A.1: The results of the preliminary user study: positive numbers our votes for our system, while negative numbers are votes for the current system.

Session	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Participant 1	-2	-1	2	1	2	-1	1	-1	0	-1	-2	-1	-2	-1	-2
Participant 2	1	-1	2	-1	1	-2	1	-2	0	-1	-2	-1	-1	0	-1
Participant 3	-2	0	2	-2	2	-2	-1	-2	0	-2	-1	-2	0	-1	-2
Participant 4	-2	-2	1	-1	2	-2	1	-1	-2	-1	-2	-1	-2	-1	-2
Participant 5	1	1	1	1	2	1	-1	-2	-1	-2	1	-2	0	-2	0
Participant 6	-1	-2	2	2	2	1	2	1	-2	-1	-1	-2	-2	-1	-2
Participant 7	-1	-2	1	0	2	-2	2	0	1	-1	-2	-2	0	-2	-2
Participant 8	-1	0	2	0	1	0	2	-1	-1	-1	-2	-1	-2	-1	-2
Participant 9	-2	-1	2	-1	2	-1	2	-1	-1	-2	-2	-1	-2	-1	-2
Participant 10	0	-1	2	0	2	-1	1	-2	0	-2	-1	-1	-1	-1	-1
Participant 11	-2	-1	2	-1	1	-2	2	0	0	-2	-1	-1	-2	-1	-1
Participant 12	1	-1	2	1	0	-1	2	-1	-1	1	-1	-1	-2	-1	-2

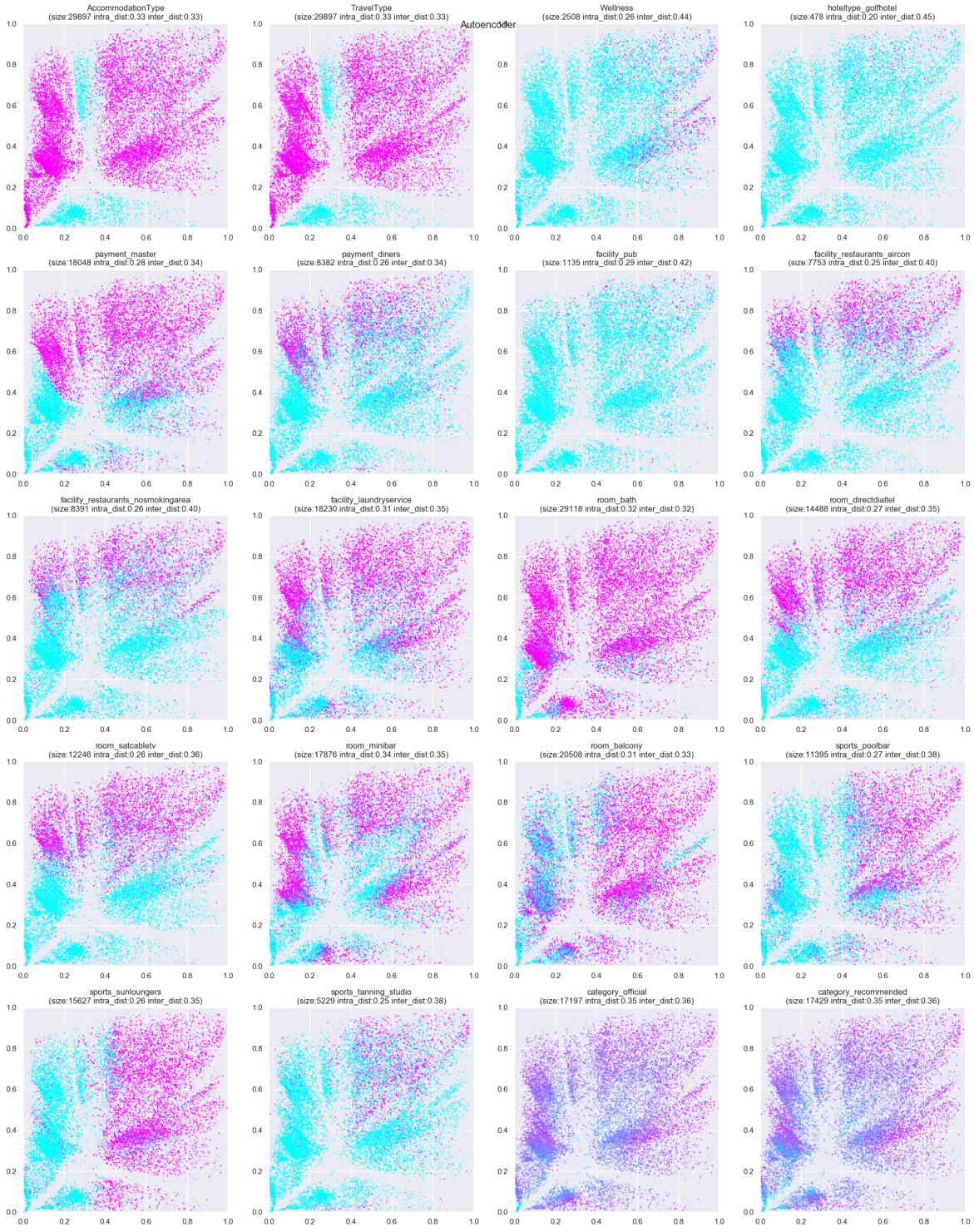


Figure A.1: The products plotted in the embedding learned using an autoencoder. Clusters are plotted for some of the features. Purple means the value of the given attribute of the product is above average and blue below average.

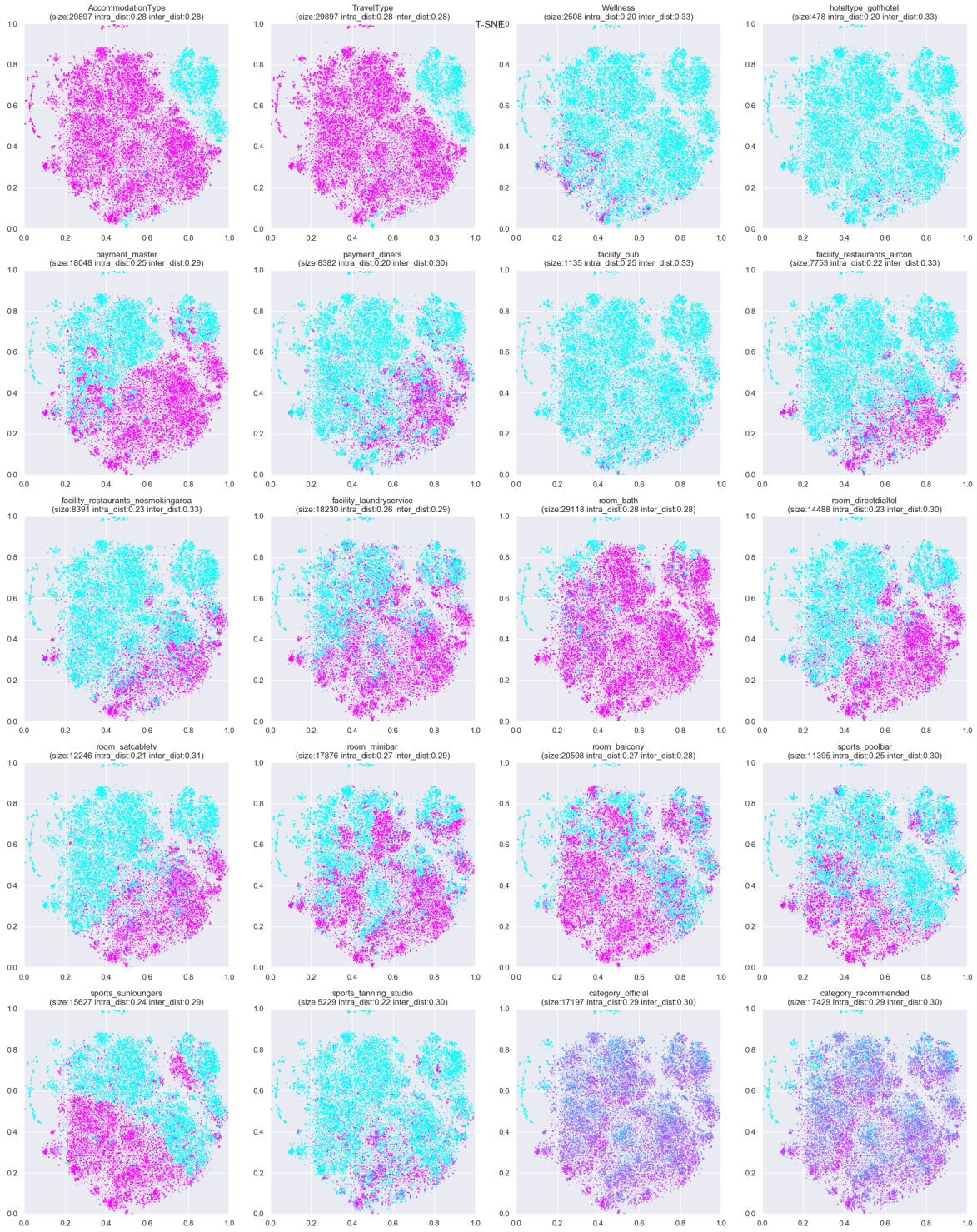


Figure A.2: The products plotted in the embedding learned using t-SNE. Clusters are plotted for some of the features. Purple means the value of the given attribute of the product is above average and blue below average.

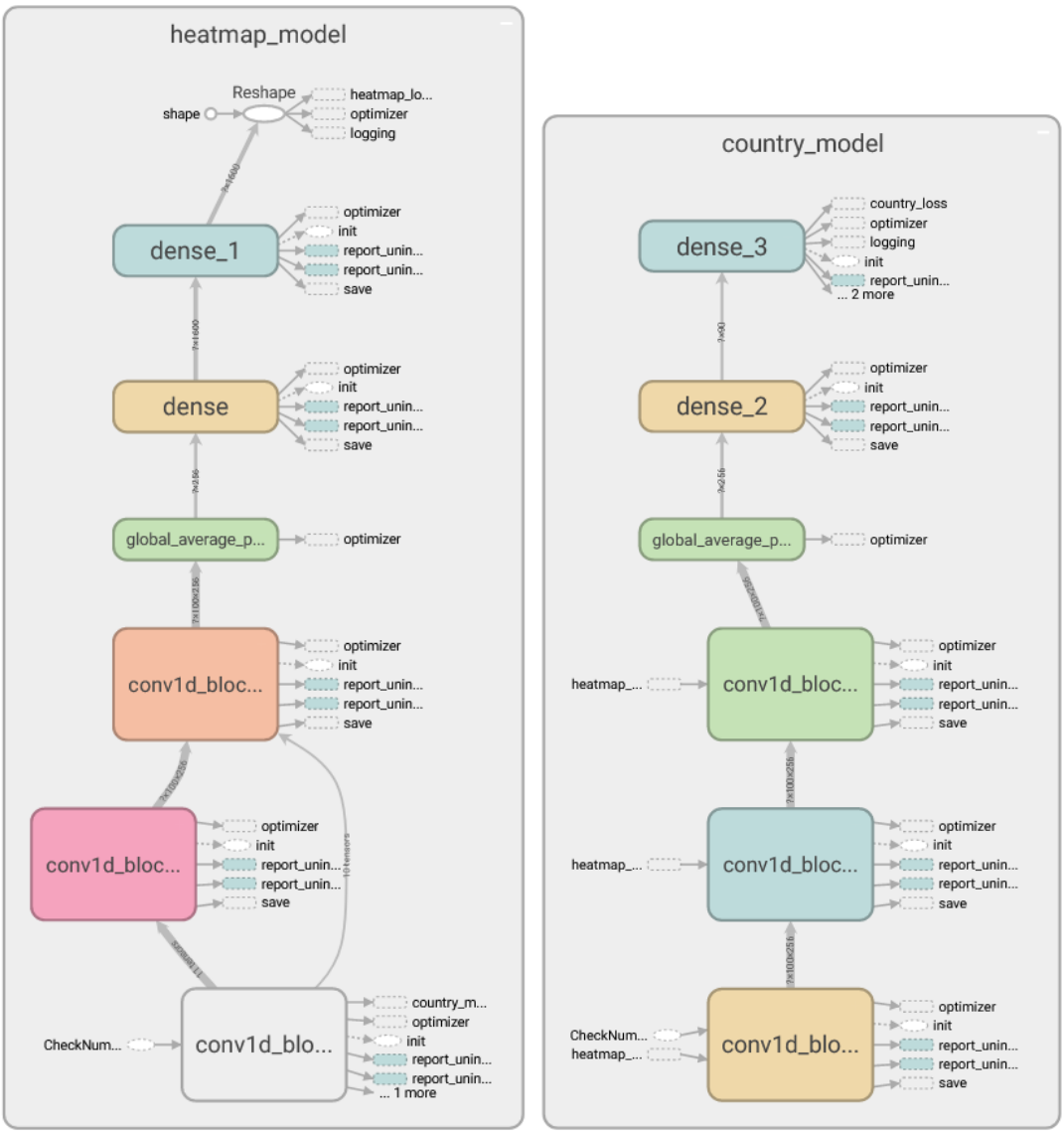


Figure A.3: The left image is an overview of the layers that make up the heatmap model, while the right is of the country model. Input comes in from the bottom, the prediction comes out of the top most layer.

Bibliography

[1] C. A. Gomez-Uribe and N. Hunt, *The netflix recommender system: Algorithms, business value, and innovation,* ACM Transactions on Management Information Systems (TMIS) **6**, 13 (2016).

[2] T. Vermeulen, J. Bijl, M. Rooijackers, K. van der Most, N. Heerschap, L. Schreven, J. Heslinga, K. van Wijk, G. Pepels, D. Nijs, *et al.*, *Trendrapport toerisme, recreatie en vrije tijd 2018*, Trendrapport toerisme, recreatie en vrije tijd (NRIT Media, 2018).

[3] A. Schulz, *The role of global computer reservation systems in the travel industry today and in the future,* Newsletter Competence Center Electronic Markets **6**, 17 (1996).

[4] D. Buhalis and R. Law, *Progress in information technology and tourism management: 20 years on and 10 years after the internet—the state of etourism research,* Tourism management **29**, 609 (2008).

[5] Z. Xiang, V. P. Magnini, and D. R. Fesenmaier, *Information technology and consumer behavior in travel and tourism: Insights from travel planning using the internet,* Journal of Retailing and Consumer Services **22**, 244 (2015).

[6] N. H. Marketing, *Kerncijfers 2018: gastvrijheidseconomie* (NBTC Holland Marketing, 2018).

[7] J. Neidhardt, T. Kuflik, and W. Wörndl, *Special section on recommender systems in tourism,* (2018).

[8] H. Werthner and F. Ricci, *E-commerce and tourism,* Communications of the ACM **47**, 101 (2004).

[9] I. Portugal, P. Alencar, and D. Cowan, *The use of machine learning algorithms in recommender systems: a systematic review,* Expert Systems with Applications (2017).

[10] F. Ricci, *Travel recommender systems,* IEEE Intelligent Systems **17**, 55 (2002).

[11] Y. Wang, S. C.-F. Chan, and G. Ngai, *Applicability of demographic recommender system to tourist attractions: a case study on trip advisor,* in *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 03* (IEEE Computer Society, 2012) pp. 97–101.

[12] J. Borràs, A. Moreno, and A. Valls, *Intelligent tourism recommender systems: A survey,* Expert Systems with Applications **41**, 7370 (2014).

[13] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, *Methods and metrics for cold-start recommendations,* in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval* (ACM, 2002) pp. 253–260.

[14] R. Burke, *Hybrid web recommender systems,* in *The adaptive web* (Springer, 2007) pp. 377–408.

[15] S. Gupta and V. S. Dixit, *Scalable online product recommendation engine based on implicit feature extraction domain,* Journal of Intelligent & Fuzzy Systems **34**, 1503 (2018).

[16] Y. S. Kim, B.-J. Yum, J. Song, and S. M. Kim, *Development of a recommender system based on navigational and behavioral patterns of customers in e-commerce sites,* Expert Systems with Applications **28**, 381 (2005).

[17] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, *Evaluating collaborative filtering recommender systems,* ACM Transactions on Information Systems (TOIS) **22**, 5 (2004).

[18] G. Adomavicius and A. Tuzhilin, *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,* IEEE Transactions on Knowledge & Data Engineering , 734 (2005).

[19] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives,* IEEE transactions on pattern analysis and machine intelligence **35**, 1798 (2013).

[20] Y. Li, M. Yang, and Z. M. Zhang, *A survey of multi-view representation learning,* IEEE Transactions on Knowledge and Data Engineering (2018).

[21] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, *Multimodal deep learning,* in *Proceedings of the 28th international conference on machine learning (ICML-11)* (2011) pp. 689–696.

[22] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity,* The bulletin of mathematical biophysics **5**, 115 (1943).

[23] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review **65**, 386 (1958).

[24] A. Karpathy, *The unreasonable effectiveness of recurrent neural networks,* Andrej Karpathy blog (2015).

[25] C. Olah, *Understanding lstm networks,* (2015).

[26] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation,* arXiv preprint arXiv:1406.1078 (2014).

[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems,* (2015), software available from tensorflow.org.

[28] L. v. d. Maaten and G. Hinton, *Visualizing data using t-sne,* Journal of machine learning research **9**, 2579 (2008).

[29] 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.