UTRECHT UNIVERSITY

DEPARTMENT OF INFORMATION AND COMPUTER SCIENCE

DECISION SUPPORT SYSTEMS

MASTER THESIS

Deep Learning Model for Clustering Heterogeneous Data

Case Study on Recognizing Booking Behavior for a major Airline

Author:
Steven BOTH

Supervisor:
Dr. Cassio DE CAMPOS

July 22, 2019

Abstract

In the world of revenue management, determining the right price for the right customer at the right time is an ongoing challenge. Within the airline industry, inventory analysts are continuously updating ticket prices to counteract the change in demand and willingness to pay. Some have to actively do this for as much as 20,000 flights. A model that could support them in their decision making by grouping similar flights would save a lot of time and effort. For this, we propose a novel clustering approach using Sum-Product Networks (SPNs) implemented as an Expectation Maximization algorithm that can handle heterogeneous data. To handle categorical variables correctly, we present our adapted version of CLARA, namely CATCLARA. For this approach, we will research what parameters will result in the optimal outcome. After validating the effectiveness of the model, we will perform a case study where we will cluster flights based on their booking behavior. The results show that an SPN with CATCLARA start yields better results than CLARA on most datasets. The outcome of the case study is that our model can correctly cluster flights based on their booking behavior, mainly by clustering flights with deviating behavior. Though the results are clearly not perfect, the model is a valuable first step towards supporting analysts in their decision making.

Key words: Sum-Product Networks, Expectation Maximization, clustering, categorical variables, CLARA, revenue management, booking curves

Acknowledgement

Throughout the writing of this master thesis I have received a great deal of support and assistance. I would first like to thank my thesis supervisor Dr. Cassio De Campos for his useful comments, ideas and engagement through the learning process of this master thesis. I am grateful to him for sharing his expertise, for the time he invested in this research and for inspiring me with his scientific enthusiasm.

Furthermore, I would like to thank Prof. Dr. Ir. Linda van der Gaag for her guidance and advice during the literature study and Dr. Silja Renooij for being the second supervisor of this thesis.

I would also like to acknowledge my colleagues at KLM for making my internship possible and for providing me with all the necessary facilities and data for the research. Special thanks goes to Lotte Adema MSc for granting me her domain knowledge and subsequently for evaluating the results.

Finally, I must express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Thank you.

But above all, I want to thank God for providing me this opportunity and granting me the capability to proceed, persevere and complete it satisfactorily. It are His care and blessing without which this accomplishment would not have been possible.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Current situation and motivation

This thesis was written as part of an internship at a major Dutch airline. Therefore, there is not only a scientific motivation to the thesis, but also a very clear business motivation. To understand this motivation, we will first give an overview of the current situation of the business, followed by both motivations.

1.1.1 Current situation

Ticket prices for flights are determined by so called revenue management strategies. Applying such strategies is done by inventory analysts. These analysts all have the responsibility for a certain group of flights (a complex), usually to the same geographical destinations, like Spain/Portugal, the Nordics or South East Asia. Some of these destinations have multiple flights every day for a whole year, meaning that an analyst sometimes has to actively keep up with as much as 20,000 flights. The difficulty is that even though these flights have similar destinations, the type of passengers and their booking behavior can completely differ. Therefore, it is not just applying the same strategy 20,000 times over and over again, but determining which strategy has to be applied to each individual flight. To determine which strategy is viable for which flight, they analyze the booking behavior of each flight and try to match that pattern to other flights that have already flown. Because comparing each individual flight to all previous flights is practically impossible to do manually, analysts use simple, rule-based analysis applications. In these applications, a flight is presented next to the results of that of the same flight in the past years. Also, there is a prediction available of the final load factor based on the same flight a year ago and the two weeks before and after that, so five flights in total. The comparison is then done manually.

1.1.2 Business motivation

This way of working has a number of disadvantages for the airline. To start with, the comparison has to be done manually. That an analysts has to act manually upon the results of the comparison is logical, because the system is only set up as a decision support system. But as with all things manual, it has a lot of space for interpretation and is thus prone to human error. Especially for new analysts who do not have as much experience with the analysis of booking behavior, a lot of revenue can be lost before they develop enough skill to effectively do this analysis, spillage as well as spoilage. A system that would take over this manual process of comparing booking behavior of flights and finding similar flights, and use these results to support the analysts in their decisions more thoroughgoing by showing which flights deviate from their expected behavior, would leave the analyst with more time to interpret the outcome and act upon it.

Secondly, even though it seems to work good enough in the current situation, it is not always a given that the chosen flights are representative as comparison for the

booking behavior of the flight. This is especially the case for dates approaching holidays or events like Christmas or Easter. Such events distort the normal booking behavior and when these flights are used as comparison for non-event flights, the comparison is also distorted. This can lead to an analyst applying the wrong strategy to a flight and thus losing revenue. On top of that, there are probably other flights that could be used as a better comparison. If flights are chosen because of their similarity in booking behavior and not because of a fixed date, this will result in a much more flexible comparison. These flights can come from the same complex of which an analyst is responsible, but also from flights to other destinations. But because analysts in the current situation do not have a complete overview over all flight, but just over their own, these overarching patterns are not easily recognized.

A system that could use any given list of flights for the comparison of booking behaviors, independently of a fixed date or a specific complex, would result in a more reliable outcome and a more accurate comparison. Because of these overarching patterns, the bigger picture will become more visible. The knowledge of this bigger picture of the flights, or more specifically, their booking behavior, will lead to a better understanding of the optimal strategy to minimize spillage as well as spoilage. And utilizing this, it could result in the generalization of steering for groups of similar flights or even for certain time slots with similar behavior.

1.1.3 Scientific motivation

Moreover, this subject is also an interesting area for scientific reasons. While for the airline only the final results matter, getting from data to results has also importance. Clustering has been an active field of study and among others, clustering heterogeneous data still poses enough challenges.

To the best of our knowledge, a novel approach of addressing these difficulties posed by heterogeneous data, is applying a deep learning classifier to this clustering problem, namely a Sum-Product Network (or SPN). For this clustering we can need an alternative clustering method by regarding the cluster labels as missing values. This missing values can be filled in using a data imputation technique, by applying Expectation Maximization (or EM). This method should be a well performing manner to deal with heterogeneous data while maintaining a low computational complexity. It should also be fairly simple to perform a sensitivity analysis on the results.

Secondly, it is interesting to understand the practical use of this clustering approach to the clustering of flights based on their booking behavior. In particular it is interesting to see if this clustering can also be used to detect flights that deviate from their expected behavior. This could be done by defining 'sibling' flights, that is, flights that have a similar profile and should exhibit similar behavior. This would be interesting from scientific as well as business standpoint.

1.2 Research question

The problem statement and motivation discussed in the previous section lead us to the following research question:

> Is it possible to use a Sum-Product Network for clustering and analysing the booking behavior of flights?

This question covers the whole scope of this thesis, and satisfies both the business and scientific motivation. To specify certain parts of the research even further, we will

formulate more specific sub research questions in section 3.5, after a thorough study of the related literature.

1.3 Outline

In chapter 2, we will give a short introduction to the field of Revenue Management. After that, in chapter 3, there is a description of the techniques and models we will use for the research as well as an overview of the previous work on these topics. This chapter will conclude with the formulation of the sub research questions. In chapter 4, we will discuss how we implemented the model to be applied for clustering. In the next chapter, chapter 5, we will describe the experiments that we conducted on test datasets to determine the effectiveness of the model and report our results. Subsequently, chapter 6 is about the case study we performed by applying the model to real world data. We will first describe the booking data and our methodology, then a review of the results, followed by the evaluation of the results and our conclusions. Finally, in chapter 7, we will answer our research questions, draw our final conclusions and discuss possible future work.

2 Revenue Management

Revenue management (RM) is a strategy that tries to maximize the revenue of a certain product by analyzing consumer behavior and predicting the expected demand for that product. The way revenue management tries to maximize the revenue is by selling similar products at different fares and controlling how, when and to who the product is sold for which fare. This can depend on the type of customer, the period of booking and how early in advance the product was booked (Weatherford and Belobaba, 2002). The airline industry was the first to apply such strategies, but other transportation industries as well as the hotel branch have incorporated revenue management in their business strategies (Weatherford, 2015). For a hotel, the products are rooms, for a car rental, they are cars and for airlines, they are the seats on their flights. Because all of these products have a limited, fixed availability, the profit on every single product counts. The key is to sell the right product to the right customer at the right time for the right price (McGill and Van Ryzin, 1999).

For any company, to be able to sell the product for the right price, it is important to know the demand for those products. The demand is the number of customers interested in buying the product. What makes it difficult is that this demand is not known. Even when historical booking data for a product is available, this is not the same as the demand. This is because there may have been customers interested in buying that certain product but due to revenue management decisions about prices or capacity, they were not able to buy the product (Weatherford and Pölt, 2002). These customers add up to the demand, but are not visible in (historical) booking data. For this reason, the demand can only be approximated by prediction. The accuracy of this prediction has a large impact on the final profit as a forecast error of 25% could result in a negative revenue impact of 0.5-2% of the total profit (Weatherford and Belobaba, 2002).

The second part of this prediction is determining the willingness to pay for each expected customer. The willingness to pay is the maximum price at which a customer is willing or able to book the product (Hinterhuber and Liozu, 2017). This willingness mostly depends on the financial situation of the customer and their travel motif.

2.1 Steering

This prediction of the demand and the willingness to pay is an ongoing process. Analysts in charge of a flight have to update their predictions continuously. Around 360 days before the planned departure date, a flight is added to the schedule. As of this moment, customers can start booking seats on that flight and at the same moment, the steering of that flight begins. When a flight is steered, it means that the prices at which a ticket can be booked are adjusted. This can be done automatically by a RM system or manually by an analyst. The adjusting of the prices is done by opening or closing certain subclasses for booking, depending on the prediction of the current demand and the number of seats available. Subclasses are the different fares at which the seats can be booked. To accommodate the willingness to pay for different customers, the price for each subclass gradually increases, ranging from the cheapest economy class fare to

the more expensive business class fare. This price is the minimum a customer has to pay for a seat within that subclass. If multiple subclasses within a section are open, the price of the cheapest subclass is used for the bookings and thus subclasses are always closed from cheapest to most expensive, within their respective grouping. Because of that, it means that if a business class subclass is closed, also all lower business class subclass are automatically closed, while economy class subclasses, with a lower fare, can still be open for booking.

Inventory steering in done by predicting and analyzing the demand for the flight. When the demand is high, there are probably more customers with a high willingness to pay, especially when there are not that many seats left, and thus the cheaper subclasses can be closed. When the demand is low, it is better to sell some cheaper tickets than to fly with an empty plane. However, because the demand is only a prediction, it is not always that simple. This can be demonstrated with a simple example (Irwin, 2010).

Suppose, flight X is scheduled to fly with an aircraft that has exactly 100 seats available. For the simplicity, there are only two subclasses, namely an expensive business class (C) and a cheaper economy class (M). The fare for subclass C is €500 and for subclass M, it is €100. It is known that customers in subclass C will book late, even until the day of departure itself, while subclass M customers usually start booking far in advance, also until the day of departure.

For the first case, suppose that the exact demand for both subclasses is known; there is a demand of 10 consumers for subclass C and 200 consumers for subclass M. In this case, the steering is easy. Because the revenue per seat is much higher for subclass C, as many seats as possible should be sold to customers who are willing to pay that fare. The other seats can be sold to M-class customers. The steering will result in selling 90 seats to early booking M-class customers and reserving the last 10 seats for the expected C-class customers. This means that as soon as the 90 seats are booked, the analyst closes subclass M for booking. The other 110 expected customers for subclass M are turned down, because from that moment onwards, it is only possible to book C-class seats. The total revenue is the maximum that could be achieved because the demand was perfectly handled.

In the second case, the exact demand for both subclasses is not exactly known, but there is a vague prediction available; the expected demand for subclass C is low and the expected demand for subclass M is high. In this case, steering is much harder. The analyst still wants to sell as many seats in the C-class, but the number of seats to reserve for these customers is a guess. Say, until some weeks before departure, when there are 80 bookings placed by M-class customers, the C-class customers start booking. One week before departure, there are 5 C-class and 90 M-class bookings, and thus only five available seats left. The analyst now has two options. The first option is to close subclass M and reserve those five seats for C-class customers. However, if those five C-class customers do not book those seats, the aircraft departs with empty seats. There was still demand for the M-class, so those seats could have been sold, were it not that M-class customers were unable to book because that subclass was closed. The phenomenon of revenue loss due to closing subclasses too early, resulting in empty seats, is called *spoilage*. The second option is to let subclass M open for booking. However, if those five seats are quickly booked by M-class customers while there were also five C-class customers that would have booked those seats, customers with a higher revenue per seat were denied due to capacity constraints. This phenomenon of revenue lost due to leaving subclasses open for too long, resulting in the aircraft filling up to quickly with cheaper customers and thus rejecting customers that were willing to pay more, is called *spillage* (Lohmann, 2011).

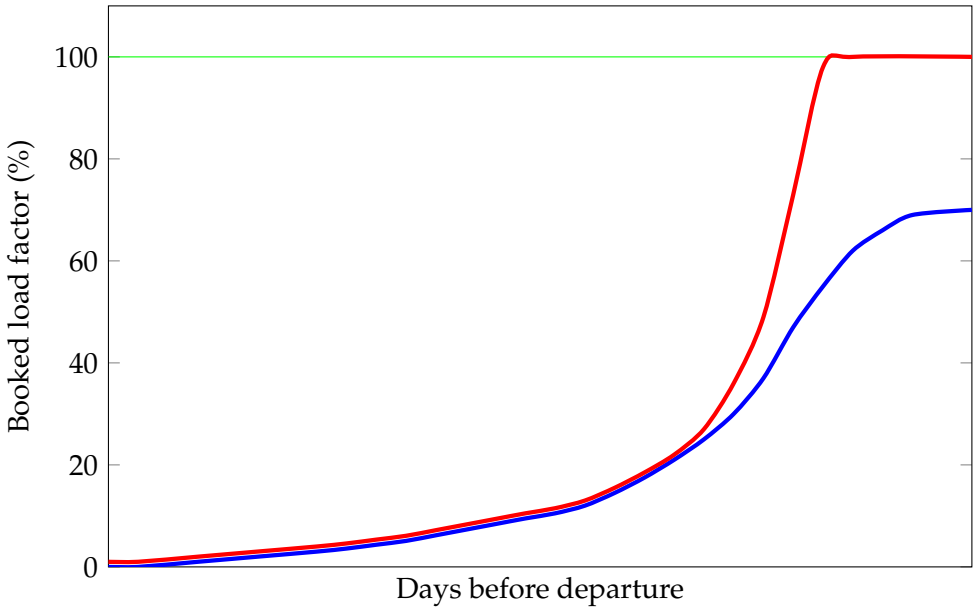

FIGURE 2.1: Booking curves (section 2.3) that show spillage or spoilage. We assume that the demand for both flights is equal. **Green**: Maximum load factor. **Red**: Seats are sold out too quickly, because they are sold for less than their potential price *(Spillage)*. **Blue**: Seats remain unsold in the presence of demand, because the prices are higher than the customers' willingness to pay *(Spoilage)*.

In general, the situation is a lot more complicated. Firstly, airlines use more than just two subclasses available to be able to accommodate various customers and their willingness to pay. Secondly, because many flights are booked in combination with a connecting flight, the steering should consider this greater scheme of flights. On one hand, this makes steering easier, because closing subclasses is no longer binary and thus the difference in price between every step becomes more gradually. On the other hand, there are now a lot more factors that have to be taken into account, making the steering more complex.

Making this process even more complex is, that because no flight is exactly the same, the demand and willingness to pay of these flights are also different. For the analyst, this results in a complicated optimization problem with a lot of unknown factors. When the bookings come in too fast (i.e. they rocket) compared to similar flights, the number of customers with that willingness to pay is apparently large enough (see RED in figure 2.1). This means that the prices could be increased to minimize the potential spillage. On the other hand, when the bookings come in too slow (i.e. they freeze), the number of customers with that willingness to pay is apparently too small (see BLUE in figure 2.1). Now, the prices should be decreased to minimize the potential spoilage. The analyst has to find an optimal strategy that minimizes both spillage and spoilage for the current flight.

On top of that, another revenue management strategy that is a complete optimization problem in its own right and worth the mentioning, is called overbooking. Overbooking accounts for all customers that book a seat but do not show up at departure or cancel just before departure (Irwin, 2010). Because of this, the aircraft would still depart with empty seats on board. The number of no-shows, as these first type of customers are called, are analyzed and analysts can decide to oversell a number of seats on their flights. These seats are thus sold twice and as a result, more seats can

be booked than there are actual, physical seats on the aircraft (McGill and Van Ryzin, 1999). This will definitely increase the revenue, because there are more products that can be sold. However, this strategy also has the very clear danger that when there are more seats sold twice than there are actual no-shows, there are more customers than seats for that flight. This will to result in customers being denied boarding, whom will have to be compensated financially for this. Therefore, the objective of overbooking is to find an optimal balance between maximizing the expected revenue and minimizing the risk of denied boardings.

2.2 Anomalies

Nevertheless, there will always be variables that are hard to be accounted for. This can include changes in the market because of competition but also unexpected events. These unexpected changes have a large impact on the demand of certain flights and can result in spoilage as well as spillage.

As mentioned before, a simple rule of thumb for inventory steering is that when the demand is high, the willingness to pay is also high. This on its turn will result in the cheaper subclasses being closed, thus raising the prices for tickets. This is especially the case during special events, like holidays, popular sport events or major business conferences. On the flights that are connected to these events, the demand is extremely high. In addition, because many people are interested in buying a ticket and they all have the same travel motive to specifically visit that event, none of them wants to miss that specific flight at that specific time to that specific place. Thus, their willingness to pay for that specific seat is extremely high. If such an event is not expected, there is not much reason to treat that flight as a high demand flight and thus it will not be steered as such. This will result in a very high spillage, even when an analyst in the final stages tries to control the damage.

This is not a problem for all events because some are expected. Special events like Christmas recur every year and always at the same date. It becomes more complex for yearly events that do not have the exact same date every year, like Easter, because now predicting the expected demand for those flights using historical data is not possible without first correcting for the shift in date. The same holds for yearly recurring events that have the same date every year but that are organized in a different place every year. Finally, there are events without any fixed re-occurrence. If these events are well known like football matches or the Olympic Games, they can be manually tracked, but otherwise they disappear in the mass, until they are discovered by an analyst, but at that point, at least some revenue damage has already been done.

2.3 Booking curve

The most basic data available is the booking curve. This curve consists of the accumulated number of bookings for a flight at each day until its departure. This curve starts when the flight is opened for booking and ends on the day of departure, thus containing approximately 360 data points, or snapshots (Irwin, 2010). The final snapshot, thus the snapshot of the day of departure, contains the last known number of bookings. This does not have to be the same number of customers that have actually boarded the flight. Last-minute cancellations, no-shows or denied boardings are not included in this data. These differences are also interesting for revenue management, but not within the scope of this research, and will thus not be taken into account.

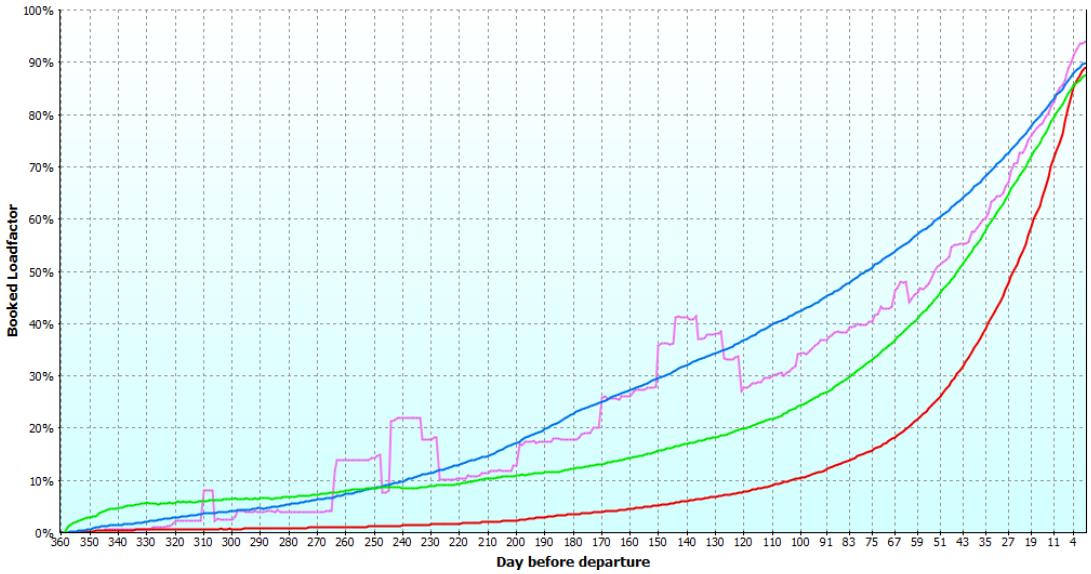

FIGURE 2.2: Four examples of booking curves, all with a different behavior.

Because the capacity can differ for each flight, it is hard to compare booking curves for different flights. To normalize the curves for all flights, the booked load factor (in short, BLF) is used. This is the percentage of booked seats relative to the capacity. Because of overbooking, this percentage can be larger than 100%.

In figure 2.2, four examples of booking curves are shown. For the red, green and blue curves, it is clear that they exhibit different behavior in terms of shape and pattern (Lee et al., 2005). The red curve shows a late booking behavior, while the blue curve shows more early bookings. The green curve on the other hand has some very early bookings, but then stagnates for a while before the next bookings arrive. Nevertheless, these three different curves all end around the same BLF. In this case this more accidental than a rule, but it shows that behavior is not the only factor determining the final load factor. Which could lead to the assumption that the final load factor is also part of the behavior, opposed to other researches that concluded that there is a clear relation between the final load factor and the shape of the booking curve (Ma et al., 2014). There it is applied to a railway company where the seats were available for booking only 12 days before departure, while the booking window for an airline is around 30 times larger, thus possibly changing the relation between the behavior and the load factor.

The fourth curve (purple) has a lot more detail than the other three curves. This is because the other three curves are aggregations over larger sets of flights, while the purple curve is the curve of a single flight. This curve fluctuates a lot more and has many steep edges, making it more difficult to find a clear behavior.

This representation and the analysis of the booking curve leaves room for a possible distortions of the behavior. Firstly, the normalization from bookings to BLF completely depends on the capacity of the flight. This capacity is mostly fixed, but is sometimes subject to change. When this happens, the BLF also changes even though the number of bookings did not change. Another problem are group bookings, which are special bookings for groups of at least 10 people. The rules are slightly different than for individuals. For example, groups have a certain time limit before which they can cancel their booking for free. This often results in early group bookings, many of which are cancelled shortly after (Svrcek, 1992).

3 Related Work

A way to support a Revenue Management analyst in finding the optimal strategy to minimize spillage as well as spoilage, is to provide a better understanding of the overarching patterns between the flights, or more specifically, their booking behavior. Our approach is that of finding flights with similar booking behavior. This could be done by means of clustering the flights into clusters with similar behavior. For this clustering we will use an alternative method by regarding the cluster labels as missing values. This missing value can be filled in using a data imputation technique. To perform this data imputation, we will use an adapted implementation of a Sum-Product Network. This model can be used because of the Expectation Maximization property of the inference process.

In this section, we will discuss the relevant technical subjects for this thesis. To start with, we will define some necessary notations and definitions in section 3.1. In section 3.2, we will explain the architectural and functional characteristics of a Sum-Product Network. On top of that, we shall discuss some methods for learning and inferring and give an overview of what research and implementations have already been done. Then in section 3.3, we will describe the process of data imputation. We shall also name some of the most widely used methods for this, but we will mainly focus on the Expectation Maximization approach. Finally, in section 3.4, we will discuss the principle of clustering and some challenges within this subject, including dealing with heterogeneous data and finding the optimal number of clusters.

3.1 Preliminaries

To be able to define the necessary techniques, models and methods, let us define some basic, formal notation. First, integers are represented with lower case letters (for example, i, j, w or v), and sets of integers with calligraphic capital letters (for example, $\mathcal{V}$ or $\mathcal{E}$). A random variables is denoted by an X in combination with a subscript (for example, X_i or X_1). A collection of these random variables is denoted by simply X. When the index of this collection (which set is defined as $\mathcal{V}$) is needed, this is denoted as $X_\mathcal{V} = \{X_i : i \in \mathcal{V}\}$. A realization of collection $X_\mathcal{V}$ can be denoted as $X_\mathcal{V} = x_\mathcal{V}$ or $X_\mathcal{V} = z$. The set of all realizations of collection $X_\mathcal{V}$ is written as $x_\mathcal{V} \in X_\mathcal{V}$ (Mauá et al., 2018). The realizations of collection $X_\mathcal{V}$ together are called 'evidence' and is denoted by $X_\mathcal{V} = e$.

When a random variable X_i is discrete or categorical, it can only have a finite possible values, taking values in $\{0, \ldots, |X_i| - 1\}$ (Conaty et al., 2018). A continuous random variable X_i can take infinite possible values, namely $\mathbb{R}$, but can be translated (by normalizing) to a value between 0 and 1.

3.2 Sum-Product Networks

Sum-Product Networks were introduced in 2011 as a new kind of deep architecture by Poon and Domingos (2011). An SPN is a directed acyclic graph, with sums and

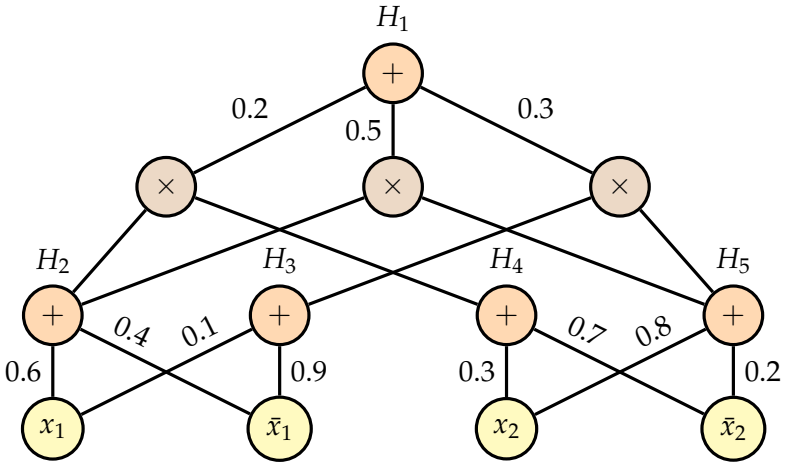

FIGURE 3.1: A Sum-Product Network over two binary variables X_1 and X_2 (Mauá et al., 2018)

products as internal nodes, variables as leaves and weighted edges connecting them (see figure 3.1). The value for each node is calculated recursively from the values of its children. The value of a sum node i is equal to

$$\sum_{j \in Ch(i)} w_{ij} * v_j$$

where $Ch(i)$ are the children of i, w_{ij} is the weight of the edge between nodes i and j and v_j is the value of node j. The value of a product node i is equal to

$$\prod_{j \in Ch(i)} v_j$$

The value of a leaf node for a discrete variable is its corresponding probability mass function (PMF), for a continuous variable this is a probability density function (PDF). The difference is that the density is not normalized to sum up to 1. Normalizing is sometimes possible, but continuous variables can have infinite many values, making normalizing usually impossible. These densities, however, are still proportional in size to each other and can thus be compared.

The value of the SPN is the value of the root node of the network. Even though there can be multiple root nodes, in this thesis we assume that an SPN only has a single. We also assume for simplicity that in this thesis, sum and product nodes are arranged in alternating layers within the network, as this can be done without the loss of any generality or vital property. The height of a node is equal to the number of edges between that node and the root node, being 0 for the root node itself.

Every edge connecting a sum node with one of its children has a non-negative weight. For a sum node, the weights of these edges have to sum up to 1, resulting in the following property:

$$\sum_{j \in Ch(i)} w_{ij} = 1 \quad \forall i \in \sigma$$

where σ is the set of sum nodes in the network. The weights of all other edges are of no importance and are thus ignored.

Because of this structure, inference can be done in linear time in their size when completeness and consistency hold for the SPN. An SPN is *complete* if for all sum nodes holds that all their child nodes have exactly the same scope, that is, they cover

the same variables. An SPN is *consistent* if for all product nodes holds that no variable is negated in one child node and non-negated in another. It has been argued that this consistency can be relaxed to decomposability, being a weaker condition, while still allowing for exact inference (Peharz, 2015; Peharz et al., 2015). An SPN is *decomposable* if no variable appears in multiple children of a product node, so at most in one child.

In the literature SPNs were firstly used for image completion. As an example of their effectiveness, they were tasked with restoring not just small occlusions but complete halves of images (Poon and Domingos, 2011). The key factor for this image completion is detecting the underlying structure of the image and SPNs offer a deep network that can effectively represent this structure. When improving certain steps of the learning process or when adding specific properties to the architecture, researchers mostly redo this original experiment to proof their increased accuracy, resulting in most papers applying SPNs to this specific field. Other fields that make use of this efficient representation of complex structures, most of which image processing related, are optical character recognition (Mauá et al., 2017; Ratajczak et al., 2014) and object classification (Luo et al., 2013; Sguerra and Cozman, 2016), but also language modeling (Cheng et al., 2014).

3.2.1 Learning

When Poon and Domingos (2011) compared their new deep architecture with other (deep) learning techniques, namely Principal Component Analysis, Deep Boltzmann Machine, Deep Belief Network and Nearest Neighbor, on the exercise of image completion, SPN seemed to have multiple advantages, not just theoretically but also in practice. It is at least an order of magnitude faster in learning and in inference than the other techniques, while also learning the network much more efficient. For this research, the SPN was learned by generating a generic dense architecture by taking random subsets of the variables and connecting them, according to the properties of completeness and consistency. The next step was to iteratively update the weights of the edges by inference until convergence. This was done by using hard EM. With this technique the weights are calculated by maintaining a count for all children of sum nodes, representing their importance in the maximization step and normalizing these counts afterwards. After this, removing all edges with weight zero and all parentless nodes except the root node, results in the final SPN.

Since then, more efficient ways of learning an SPN were developed. They improved upon the static architecture by using a dynamic approach by analyzing the data and exploiting the internal relations between variables or instances. This is done by clustering the variables to identify cohesion (Dennis and Ventura, 2012) or by identifying independent subsets of variables within the data (Gens and Domingos, 2013). The latter methods makes direct use of the structure of the SPN, by recursively splitting the variables into smaller SPNs. In this recursion, when variables can be split into subsets that are mutually independent, the resulting sub-SPNs are treated as children of the same product node. When this independence is not found, the instances are clustered into similar subsets where they are treated as children of the same sum node, with weighted edges relative to their size.

For this thesis, an adaption of the algorithm defined by Conaty et al. (2018) will be used for learning. The algorithm is shown below.

LEARN(D, PRODUCT-LAST, MAX-HEIGHT, HEIGHT): returns an *SPN*
Inputs: *D*: dataset; PRODUCT-LAST: if the parent-node was a product-node; MAX-HEIGHT: controls the height; HEIGHT: starts at 0 for the main root node

1. If *D* contains a single variable, then

   (a) Create a sum node S with children as the leaf nodes corresponding to the values of that variable, and weights according to their frequencies in the data.

   (b) Return S.

2. If HEIGHT equals MAX-HEIGHT-1, then

   (a) Create a product node S and partition the dataset into $D_1, \ldots, D_t$ (where t is the number of variables in D), with one single variable per D_i.

   (b) For $i = 1, \ldots, t$, call LEARN(D_i, FALSE, MAX-HEIGHT, HEIGHT$+1$) and add these $SPNs$ as children of S.

   (c) Return S.

3. If not PRODUCT-LAST, then

   (a) Create an empty product node S. Create an empty (undirected) graph.

   (b) For every i, j, compute G-TEST(D, X_i, X_i) and if the p-value is below PVAL-THRESHOLD, include an edge (i, j) in the graph.

   (c) Compute the connected components $C_1, \ldots, C_t$ of the graph, and partition the dataset D into $D_1, \ldots, D_t$ based on the variables that appear in each component (D_i shall contain all data related to variables in C_i).

   (d) If t equals 1, then break

   (e) For $i = 1, \ldots, t$, call LEARN(D_i, TRUE, MAX-HEIGHT, HEIGHT$+1$) and add each returned SPN as a child of S.

   (f) Return S.

4. Create an empty sum node S.
5. Partition the dataset D into $D_1, \ldots, D_t$, using a call to the PARTITION-AROUND-MEDOIDS clustering algorithm, where samples are seen as multi-dimensional vectors.
6. For $i = 1, \ldots, t$, call LEARN(D_i, FALSE, MAX-HEIGHT, HEIGHT$+1$) and add each returned SPN as a child of S with associated weight proportional to number of samples in cluster i.
7. Return S.

As can be seen, the algorithm will iteratively learn the network layer by layer, alternating between sum and product nodes and once a branch only contains a single variable, the values of that variable are created as leaf nodes linked together by a sum node and weighted edges.

Beside this basic architectural implementation, this algorithm also presents an extra feature that increases the performance of the SPN: a MAX-HEIGHT can be provided. When while learning the height of a node reaches this maximum height, the search for independent variables is halted and the remaining variables are forced into leaf nodes, limiting the overall height of the network.

3.2.2 Inference

After the SPN has been learned, it can be used for inference. This can be done in multiple ways, depending on how much data is present. In our case only the cluster label is unknown, but we do know how many total possibilities there are. Because of this we can loop over all possibilities and pass the known variables combined with a cluster label, on to the SPN. The SPN then uses the formulas given in section 3.2 to recursively calculate the probability for that combination of values. When this is calculated for every cluster label, intuitively, the cluster label with the highest probability is the most likely scenario.

However, in other cases it can happen that one or more variables are not known that are impossible to fill in like the cluster label. Inference can still be done and will calculate the most probable values for those missing variables. The variables that

are known are called *evidence*. Given a certain evidence e, the leaves of the SPN are updated correspondingly. Then the value for each node is calculated bottom up, but this time not with the standard formulas. Before calculating the probabilities, every sum node is turned into a max node. This max node does not sum over, but just takes the highest value from among its children. So, the value of max node i becomes equal to

$$\max_{j \in Ch(i)} w_{ij} * v_j$$

where $Ch(i)$ are the children of i, w_{ij} is the weight of the edge between nodes i and j and v_j is the value of node j.

After these calculations, the whole network is translated into a path of maximum probability. This path splits at product nodes to all its children, but at sum nodes (or max nodes) the path continues only to the child with the maximal value. The paths continue until they reach a leaf node. The leaves covering values of variables that are given in the evidence, are guaranteed to always be at the end of such a path. For the variables that are not given, the path of maximum probability can only end with one of the corresponding leaves: the leaf containing the most probable value for that variable. The final step is now to just follow the path, starting at the root node and selecting those values for the variables that were not given in the evidence.

3.3 Data imputation

It is a given fact that missing data causes a lot of difficulties in machine learning and data mining when it is not properly handled. It may generate bias and could affect the quality of the performance. Generally there are two kinds of missing data: Missing At Random (MAR) and Missing Not At Random (MNAR). When data is missing not at random, this means that there is a relationship between the missing data and the value of that variable itself (e.g. a score that is not filled in might suggest that that score was not that good). Nelwamondo et al. (2007) suggests a third kind of missing data: Missing Completely At Random (MCAR). In this case, the missing data has absolutely no dependence on any other variable in the data. It is therefore impossible to infer its value from the data.

A lot of different methods have been developed to cope with this problem, but according to Li et al. (2004) they can be roughly divided into three categories.

The first category is *ignoring and discarding data*, meaning that instances with missing values are ignored or deleted from the data. This method is only possible if there is enough data available left after deleting the incomplete instances. If there are too few instances left, this can impact the variance and mean of variables hugely, resulting in crooked statistical results. This is also only possible when the data is missing at random, because deleting the instance in the case of MNAR will result in deleting possibly valuable information.

The second category is *imputation*, which fills the missing variables in the data using plausible values based on information that is available (Zhang et al., 2018). The methods in this category range from very simple methods with an easy implementation to more complicated, but robuster methods. Some of the most common methods are (a) *mean imputation*, where a missing value is filled in with the mean of that variable, so that the sample mean of the variable will stay the same; (b) *Hot/cold deck imputation*, where the missing value is filled in with a random value of that variable from the data set. Hot means that the random value is drawn from that same data set, while cold signifies that the value came from an external data set; (c) *Regression imputation*,

where the missing value is filled in with the most likely value of that variable after comparing the other variables of the instance with missing data to the variables of the complete instances in the data set; (d) *Multiple imputation*, where the missing value is filled in with the average of the outcome of multiple single imputations. This imputation can be done using any other single imputation methods, resulting in multiple data sets where the missing data is filled in with different values. Multiple imputation combines the results of these data sets to minimize the noise otherwise induced by imputation (Li et al., 2004).

The third category of imputation methods is called *parameter estimation* and uses Expectation- Maximization techniques to estimate the most probable parameters for the missing variables. It is just this last category of methods that we will discuss in more depth in the following subsection by explaining the approach of Expectation Maximization.

3.3.1 Expectation Maximization

Expectation Maximization is an approach to calculate the maximum likelihood of variables in a statistical model when the values of not all variables are known, first introduced by Dempster et al. (1977). There has to be an interdependence between these unknown, or unobserved, variables and the known variables in the data set, otherwise inferring the values of these missing variables becomes impossible and meaningless (Nelwamondo et al., 2007).

The difference with Maximum Likelihood Estimation, even though both can find the 'best-fit' values, is that MLE first accumulates all data and than constructs the most likely model. EM on the other hand, first guesses the missing values and than iteratively improves the model to fit the guesses and the known data. It is guaranteed that the guesses (i.e. the estimations) of the missing values improves after every iteration. This is because of the fundamental steps of the EM approach. The steps are as follows:

1. Choose a initial value for the missing variables
2. The E-STEP: For each missing value, estimate the values of the missing variables and give the expected probability distribution for these estimations, using the given data
3. The M-STEP: Use the expected probabilities from Step 1 to update the missing variables, by selecting the values with the highest (or maximum) probabilities
4. Repeat Steps 2 and 3 until the model converges (i.e. a model that does not change from the E-Step to the M-Step)

Even though this algorithm improves the values of the missing variables at every step, it is possible that it gets stuck in a local maximum. This can be compensated for by restarting the algorithm with different random initial values in Step 1. From these multiple runs, you can choose the one that resulted in the greatest maximum likelihood (Moon, 1996).

The simple EM steps described above are of a so called *Hard EM*. Another variant of these steps is called *Soft EM* (Lázaro et al., 2003). As can be seen in the steps, the E-STEP of gives the expected probability of each possible value, based on the given data. The difference between these two EM approaches is how the maximization using these probabilities is done. The M-STEP of Hard EM uses the expected probabilities and selects the value with the highest probability and updates it in the data. For Soft EM, the M-STEP does not select the highest probability, but gives for all possible classes their

maximum probability (for example used by Segal et al. (2002)). These probabilities are then again used in the next E-STEP and this continues until convergence.

3.4 Clustering

The field of clustering has been an active subject for research for years, combining multiple disciplines as statistics, pattern recognition and machine learning (Berkhin, 2004). It is the formal study of algorithms and methods for grouping items according to their similarity. Opposed to classification, clustering does not tag these items with some categorical label, but tries to find the underlying structures between the items (Jain, 2010).

3.4.1 Heterogeneous data

In the past years, there has been an increasing interest in clustering all sorts of data, especially data that consists of multiple types of data, such as categorical, numerical or continuous (Abdullin and Nasraoui, 2012). However, the problem with categorical variables is that it is hard to reason that a value is similar to other values (e.g. GREEN, RED, BLUE) in the same way real numbers are similar (Ganti et al., 1999).

Because it is the simplest to cluster data when all variables are of the same type, this was the first and main focus of the research and development in this area. K-means for example, a simple clustering algorithm, is widely used for clustering numerical or binary variables, but dealing with categorical requires more effort (Huang, 1997). Clustering heterogeneous data thus requires some extra effort. In general, there are two options: adapting the data to make it suitable for the algorithm, or adapting the algorithm to make it suit the data.

Adapting the data can be done by converting all categorical variables into one format, usually binary, and then using a normal clustering algorithm. For example, the categorical variable 'day of week' would be converted into 7 binary variables: 'isSunday', 'isMonday', 'isTuesday', etc. The problem here is that a categorical variable is replaced by equally many binary variables as the number of possible values, increasing the size (and potentially the complexity) of the data set.

An option that does not require an adaptation of the data is including a well defined distance function for every categorical variables in the algorithm. Because of the nominal/ordinal property, standard distance functions (e.g. Euclidean and Manhattan distance) are meaningless for categorical variables. The simplest example of a distance function would be a function that returns a value of 1 when two variables have the same value and 0 otherwise. More complex distance functions are possible, but they often require an extra analysis of the data and the relation between values of each categorical variable, like the distance function proposed by Ahmad and Dey (2007).

3.4.2 Number of clusters

When all technical aspects of clustering are implemented, there still remains a final difficulty that needs to be overcome, namely determining the optimal number of clusters. For a lot of clustering algorithms it is required to provide the number of clusters as parameter. However, this optimal number of clusters is very subjective, because there is no clear definition of what a cluster is, nor are they clearly separated into groups (Tibshirani et al., 2001). Fortunately, over the years heuristics have been suggested to give a more objective solution.

The simplest heuristics is by looking at the total distance error. This is some sum of the distance between every data point and the centroid of the cluster it belongs to. When this error is low, the clusters fit the data points very well, so you could assume that minimizing this error should lead to the optimal number of clusters. But unfortunately, this would result in putting every data point in its own cluster, reducing the error to 0, while also undoing the complete purpose of clustering.

However, this heuristic can be useful when combined with some extra analysis, namely that it is not necessary to reduce this error completely to 0. When calculating this error for multiple values, there will come a point at which adding an extra cluster does not reduce the error much further. When plotted, at this point the graph looks like a bend arm, hence the name *Elbow (or L-) method* (Salvador and Chan, 2004). Determining this elbow can be done by plotting all points and selecting by eye, or by calculating the difference between all points and selecting the first point for which the difference is smaller than a certain threshold.

3.5 Sub questions

In order to answer the main research question stated in section 1.2, there are some smaller problems that need to be solved beforehand. Mostly coming from the studied literature, these sub research questions are as follows:

1. What adaptations or implementations are required for a Sum-Product Network to be used for clustering?
2. What combination of parameters will lead to the optimal results in the experimental setup?
3. How can the quality of the resulting clusters be validated?
4. Is it possible to detect flights with deviating behavior by looking at the clustering in combination with 'sibling' flights?

The first question covers the part of specifically adapting an SPN to use it for clustering instead of classification. The second question takes the data in more consideration and can be useful when improving the model beyond its basic performance, while it also deals with a problem that could potentially have a major impact on the results of the model, namely the optimal number of clusters for a given set of flights. The third question is not so much a technical problem, but more a research problem. Because there is no clear benchmark available to compare the results with, it is important to find another way of validating the results.

The fourth question takes into account the analysis part of the main research question. When there is enough time available, we will investigate whether the results hold more information than just simply clustering the flights.

4 Implementation

In this chapter we will describe the implementation of our model. To start with, we will discuss the steps we took to combine the SPN with the EM-algorithm that resulted in the clustering model. After that, we report the changes to the model to optimize the model, mostly in terms of speed. Finally we will further specify our implementation of the learning algorithm for the SPN and propose a novel solution.

After this, we ran experiments with the model as a test using some publicly available datasets (see Chapter 5). The whole process of building the model and running experiments on the test data has been very cyclic. This means that decisions about changes in the implementation of the model were made in interaction with some experimentation.

Implementing the clustering model was done within RStudio, version 1.1.463 with R version 3.5.2. The essential parts of the code is included in appendix C.

4.1 Expectation maximization

We started by implementing the four steps of an EM-algorithm. The first step is generating an initial distribution. For this we implemented two options. The first option randomly assigns the instances over all clusters. This will result in clusters of approximately equal size, but with a very low accuracy. The second option is already applying a simple clustering algorithm on data and using those results as initial distribution. These clusters already hold more information to start with.

The next step is the E-step. Here we implemented the learning algorithm for the SPN, using the code discussed in section 3.2.1. The SPN is then learned using the data including the assigned cluster for every instance. One difference in implementation was that we removed the functionality of setting a MAX-HEIGHT. In the first test runs it became clear that forcing nodes that reached the maximum height to split into leaf-nodes, lowered the accuracy of the model drastically. By enforcing this maximum height, sum-nodes are created with leaf-nodes that to have any predictive power, should have been split further. But because those nodes are used to calculate the probability during inference, they distort the results. Therefore we removed that option in this implementation.

Next is the M-step. First, we store the cluster labels for every instance and remove them from the data. This column is now treated as missing data. Doing inference on the SPN that was learned at the previous step, will result for every instance in the cluster label that has the highest probability. These are inserted in the data to replace the 'missing' data.

The final step checks whether the clusters have converged. If they have converged, the algorithms stops and outputs the clusters, otherwise both EM-steps are repeated until they do converge.

The resulting algorithm is shown below.

CLUSTER(D, NCLUSTERS): returns indices for clustering
Inputs: *D*: dataset; NCLUSTERS: the number of clusters

1. Generate an initial clustering distribution C over NCLUSTERS for all instances of D
2. Create dataset D_C by adding C as a column to dataset D
3. While C has not converged, then
   (a) Create an SPN S_C by calling LEARN(D_C)
   (b) For $i = 1, \ldots, n$ (where n is the number of instances of D), do inference with S_C for instance i and replace C_i by this value
   (c) Update dataset D_C with the new values for C_i
4. Return C

4.2 Convergence

Convergence is reached when the new clusters generated at the M-step, and the original clusters with which the SPN was learned in the E-step, are approximately equal. For this comparison, there is an allowed difference of $\alpha = 0.005$ or 0.5%.

The difficulty with comparing clusters is that the specific cluster label has no value, it only matters which elements are grouped together in it. Therefore by just comparing the assigned cluster, you are not guaranteed to end up with the correct results. If for example a dataset is grouped into two clusters. If for some reason, in the M-step the model assigns all elements of the first cluster to the second cluster and vice versa, the similarity between the old and new clusters is zero because the cluster labels are different for every element. But when looking at the grouping of elements, they are identical. The only difference is that the cluster labels of the new clusters are a permutation of those of the old.

To compensate for this problem, when the model checks whether the clusters have converged, it calculates the difference for every permutation of the cluster labels and selects the one with the highest accuracy.

This approach works for datasets with only a few cluster labels. But because the number of permutations for n clusters is $n!$, it looses its effectiveness for larger numbers of clusters. For these cases we used an implementation of the Hungarian algorithm (Kuhn, 1955). This algorithm takes a contingency table with the number of similarities between the cluster labels of the old and the new clusters. In polynomial time it then solves for the assignment that maximizes the accuracy between both clusters. This assignment is then used to determine the convergence.

4.3 CLARA

The learning algorithm also mentions a PARTITION-AROUND-MEDOIDS clustering algorithm. This algorithm is used for determining dependencies between instances. When an independence between two clusters of instances is found, the instances are split as children of the same sum node.

As this algorithm, we used CLARA (Clustering LARge Applications). CLARA was suggested by Kaufman and Rousseeuw (1990) in chapter 3 of their book *Finding Groups in Data: An Introduction to Cluster Analysis* as a means of handling large numbers of objects. Instead of dealing with with the entire dataset, it draws a small sample of the data and tries to find medoids for that sample. The other instances that are not in the sample are assigned to the closest medoid. To prevent a sampling bias, the algorithm repeats this sampling and clustering a certain number of times and finally selects the solution with the minimal distance between all instances and their medoid. How often this sampling is done is user-defined.

The difficulty is that CLARA can only handle numerical data, so in order to use it on categorical data, the algorithm needs to be adapted. But just translating the categories to numbers is not enough, for the distance function would still be wrong. For categorical variables that are nominal or ordinal, the distance between two options has no direct value. In terms of the distance function, the difference between Monday and Wednesday should be equal to the difference between Tuesday and Friday. The correct distance function should be that if they are equal the distance is 0 and 1 otherwise, because on data that is normalized to values between 0 and 1, this is the maximum distance.

For CLARA to handle such categorical variables, the data can be transformed by splitting the category into separate columns, as described in section 3.4.1. Now the distance is always equal to 0 or 1 for each column. But if they are not equal, the distance would be 1 for two columns (e.g. isMonday and isWednesday), so the total distance for this category is 2. We solved this by changing the value for each option depending on the distance function. CLARA in *R* can use three possible distances functions, namely Manhattan, Euclidean and Jaccard, but we decided to leave Jaccard out of the scope. For Manhattan the distances are just summed up, so by filling in 0.5 instead of 1 in the corresponding columns, the distance now sums up to 0 or 1. The Euclidean distance uses squares of the values, so here the corresponding columns are filled in with $\frac{1}{2}\sqrt{2}$. For both distance functions, the sum over all columns of a category now equals 0 or 1. We implemented this as a pre-processing step of CLARA. This resulting algorithm will be addressed as CATEGORICAL CLARA (or, in short CATCLARA).

4.4 Optimization

For the model, we optimized the learning algorithm in section 3.2.1. As an extra feature, we keep track the size of each node. The size of a node is the number of nodes in the its sub-network, including itself. The children of a node then are ordered on increasing size. In the inference process, this results in the smallest sub-networks being calculated first. The second improvement is that when during the inference a product node is found with a child with a probability of 0, this is returned to the parent without calculating the other children of that node. Because once a product node has a probability of 0, it does not matter what the probability of the other children is.

This also guarantees an optimization of the inference process as it is described in section 3.2.2. Because the first layer of the SPN splits on cluster label (i.e. the variable for which we loop over every value), only the sub-network belonging to that given cluster label is calculated. The sub-network belonging to the other cluster labels are also calculated, but a leaf node with the other cluster label (being the smallest child) is calculated first, the probability of which is 0. Because of this improvement, that whole sub-network will return a 0, without further calculations. This results in only the sub-network belonging to the given cluster label being calculated, thus speeding up the process of inference.

A second way we optimized the model was by implementing multi-threading for the inference process. However, this was only to optimize the speed of the model and has no further effect on the quality of the results.

5 Testing experiments

Before applying the model to the booking data, we first need to validate that the model is actually functional. To test this, we will run the model on various standard datasets with different parameter settings.

5.1 Data

The datasets that we use for testing all come from the UCI Machine Learning Repository[1]. They are all meant for classification problems, so they all contain class labels. This makes them very suitable for testing our clustering algorithm, because we can validate the outcomes using those class labels.

For the datasets, we selected sets with different data types. We distinguish between four different types, two being numeric (real and integer) and two categorical (binary and categorical). Some datasets only contain one type of data, but there are also mixed datasets. The characteristics of those datasets are shown in table 5.1.

5.2 Methodology

To test the model, we will run multiple experiments. The first experiment is to test if the model has a higher accuracy than just randomly assigning the clusters. We will run the model from a random initial distribution and calculate the accuracy of the outcome. We will compare this accuracy with the accuracy of the initial distribution. For this, we will do 10 runs with the model and use the mean accuracy for the comparison. Furthermore, we perform the experiments for different values of α, namely 2%, 1% and 0.1%. While learning the SPN, when it splits in multiple product-nodes, an independence test is done to find dependencies in the data. These dependencies determine how the data is split up. α is the threshold for this independence test. A change in the value for α should impact the number of dependencies that are found and should thus impact the size of the SPN.

Secondly, we will repeat this experiment, but now we run the model starting from a clustered distribution (using CATCLARA) to test if the model will improve these results any further.

Finally, we will test the effectiveness of our adaptation to the CLARA algorithm. We will do this by running both versions of CLARA on the data and then comparing the mean accuracies of both outcomes. For these experiments we use the Manhattan distance function and a sample size of 200. CLARA is not influenced by the change in α, so these experiments will only be performed once.

To summarize, this will result in running the experiment with 5 different versions of the model. The first three versions do not use the SPN and are only concerned with the starting distribution: random, CLARA and CATCLARA. These versions are therefore not dependent of α, so we will only run those once. The other two versions are the SPN with a random starting distribution and the SPN with a CATCLARA starting

[1] https://archive.ics.uci.edu/ml/index.php

Name	Instances	Variables	Numeric		Categorical		Classes
			Real	Int	Binary	Cat.	
Labor	57	16	-	-	11	5	2
Postoperative	90	8	-	1	2	5	3
Zoo	101	16	-	1	15	-	7
MB promoter	116	57	-	-	-	57	2
Bridges	122	10	-	-	5	5	6
Lymph	148	17	-	-	11	6	4
Iris	150	4	4	-	-	-	3
Hepatitis	155	19	-	-	19	-	2
Wine	178	13	11	2	-	-	3
Autos	205	25	-	2	18	5	2
Glass	214	9	9	-	-	-	6
Audiology	226	69	-	-	62	7	24
Breast cancer	286	9	-	4	3	2	2
Heart-h	294	11	-	-	7	4	5
Haberman	306	3	-	1	2	-	2
Dishonest users	322	4	-	3	-	1	2
Ecoli	336	7	7	-	-	-	8
Ionosphere	351	33	-	-	33	-	2
Colic	368	22	-	-	9	13	2
Vote	435	16	-	-	16	-	2
Cylinder	539	32	1	3	22	6	2
Credit approval	653	16	3	3	4	5	2
Diabetes	768	8	-	-	8	-	2
Solar flame	1,066	11	-	3	4	4	6
Car evaluation	1,728	6	-	-	-	6	4
Mushroom	8,124	21	-	-	3	18	2
Crowdsourced	10,546	28	28	-	-	-	6
Letter	20,000	16	-	-	16	-	26
Adult	30,162	14	-	6	1	7	2

TABLE 5.1: The UCI datasets used for testing the SPN clustering model.

distribution. For these, the value of α matters, so these version will be run thrice. In total this will result in 9 distinct runs for all datasets.

5.3 Results

The results of the experiments are shown in Appendix A. In these tables the mean accuracy over 10 runs is shown together with the standard deviation. The results for the different starting distributions are shown in table A.1. In table A.2 the results for the SPN with a random start are shown for the three different values for α, and in table A.3 the results for the SPN with a CATCLARA start, again for the different values for α. The results marked with red are significantly the best.

The results of the random starting distribution will be used as base line for the other results. For these random starting distributions, even though no dataset reached a mean accuracy above 55%, still almost halve of them have a mean accuracy between 50% and 55%. These datasets all have in common that they have two classes with

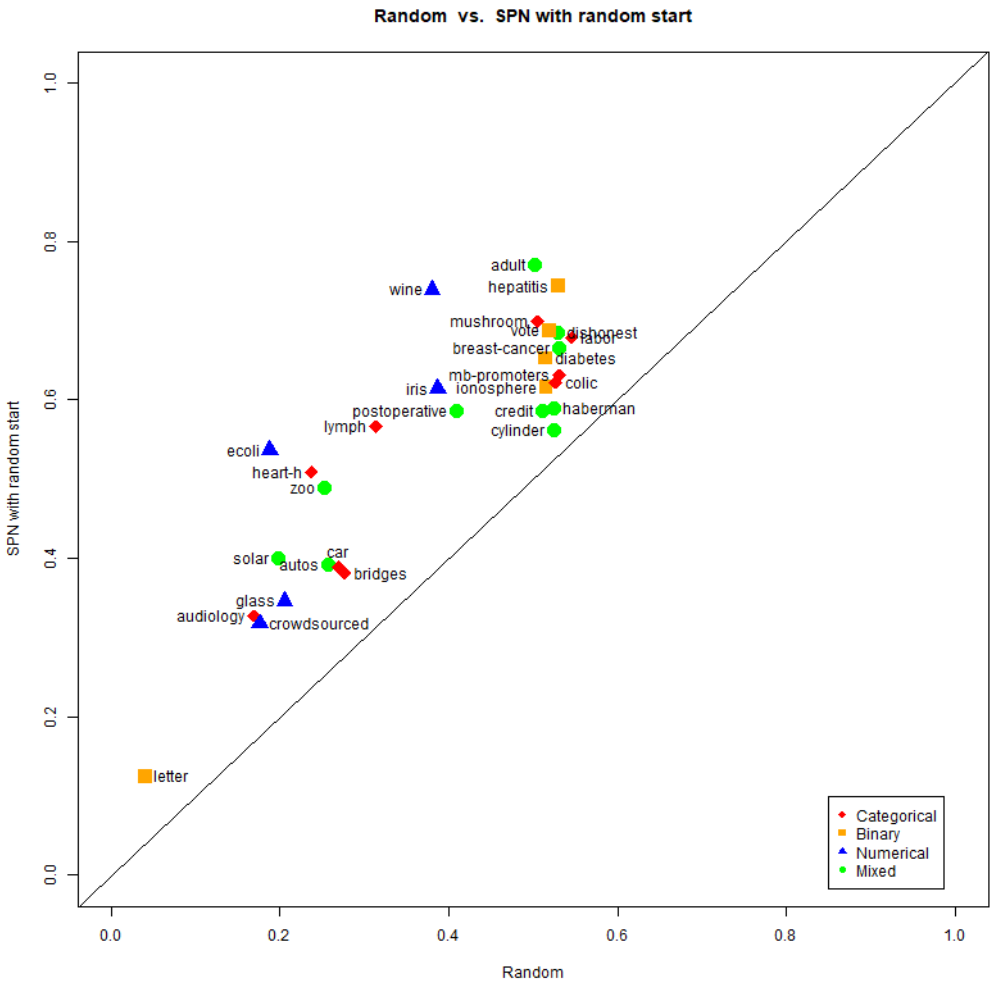

FIGURE 5.1: Results of SPN with a random start plotted against the results of random, averaged over 10 runs.

approximately equal distributions. The other dataset, most of them having more than two classes, are spread between approximately 20% and 40%.

When these random results are compared with the results of the SPN with a random starting distribution, it becomes clear that applying the SPN results in a higher mean accuracy for every dataset. Figure 5.1 clearly shows this, as all datasets are above the identity line, and thus in favor of the SPN. However, not all results are also significantly higher than the random start. This is the case for Vote, Cylinder and Credit approval. For the SPN with random start there is no significant difference between the different values for α.

Compared with the results of the random starting distribution, both CLARA as well as CATCLARA result in a significantly higher accuracy, for all but two datasets (Haberman and Breast cancer). Note that the standard deviation for both versions of CLARA is zero for all datasets. This means that all runs of the model resulted in exactly the same accuracy.

Figure 5.2 shows the results for the runs with CLARA and CATCLARA as starting distributions, plotted against each other. The results for all numerical and binary datasets are identical for both versions. For the mixed and categorical datasets, there is a lot more variance. Most results are approximately equal, but the differences are

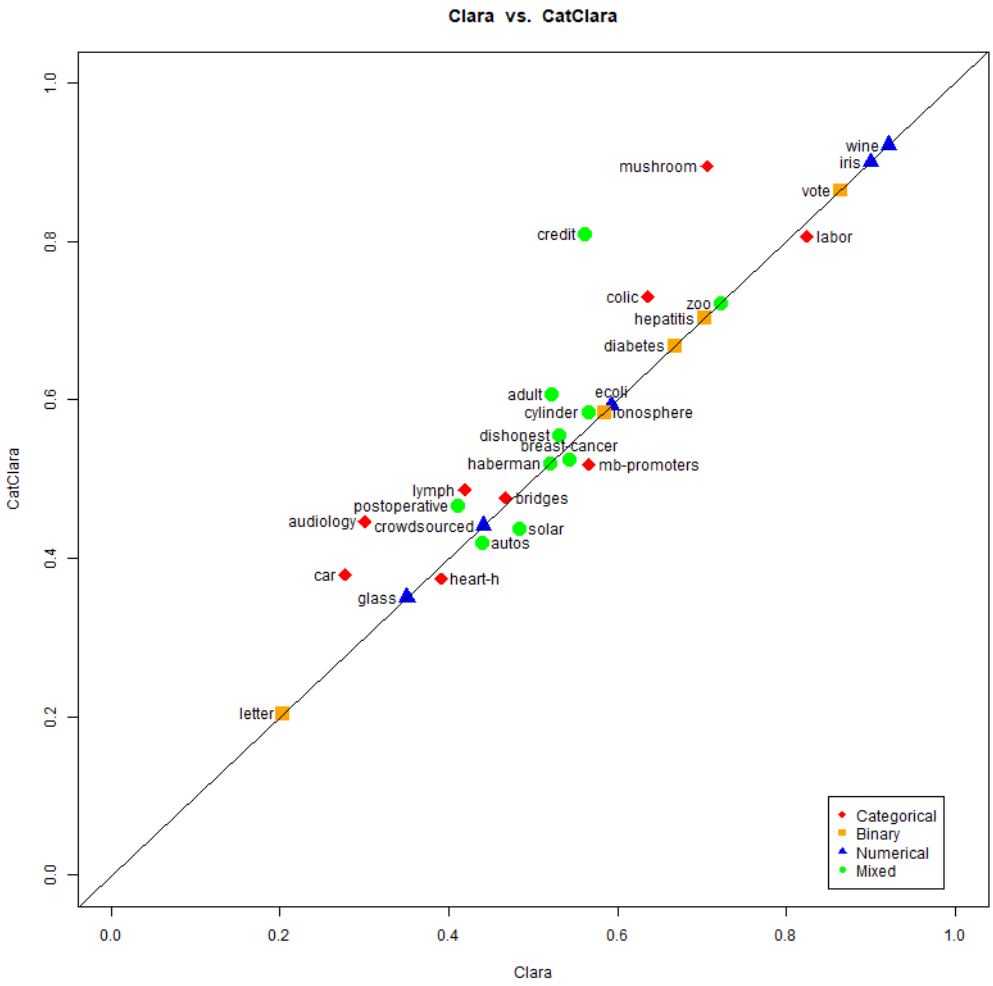

FIGURE 5.2: Results of CATCLARA plotted against the results of CLARA, averaged over 10 runs.

slightly more often in favor of CATCLARA, for the results of twenty datasets, three are equal, ten are in favor of CATCLARA and the remaining seven are in favor of CLARA. Also when looking at how far the points are from the identity line, the distances is greater for the datasets where the results of CATCLARA were the highest (e.g. Credit, Mushroom and Audiology).

Even though for most datasets the difference between the values for α is only a couple percentages, for some this differs up to 20%. These large differences are mainly between the values 0.1% and 2%. Overall, of the three values for α, 0.1% had for the most datasets the highest result. For approximately two-thirds of the datasets, applying the SPN on top of CATCLARA further improves the results. For the other datasets, the results stay the same or worsen a bit, with Solar and Colic worsening the most by 5% and 3% respectively.

Figure 5.3 again shows a comparison between CLARA and CATCLARA, but now the results of the latter have been replaced by the results of the SPN with a CATCLARA start. The results of the binary and numerical datasets were equal and still are approximately equal. But where first CATCLARA only performed better on halve of the mixed and categorical datasets, after applying the SPN, this has increased to four-fifth of the

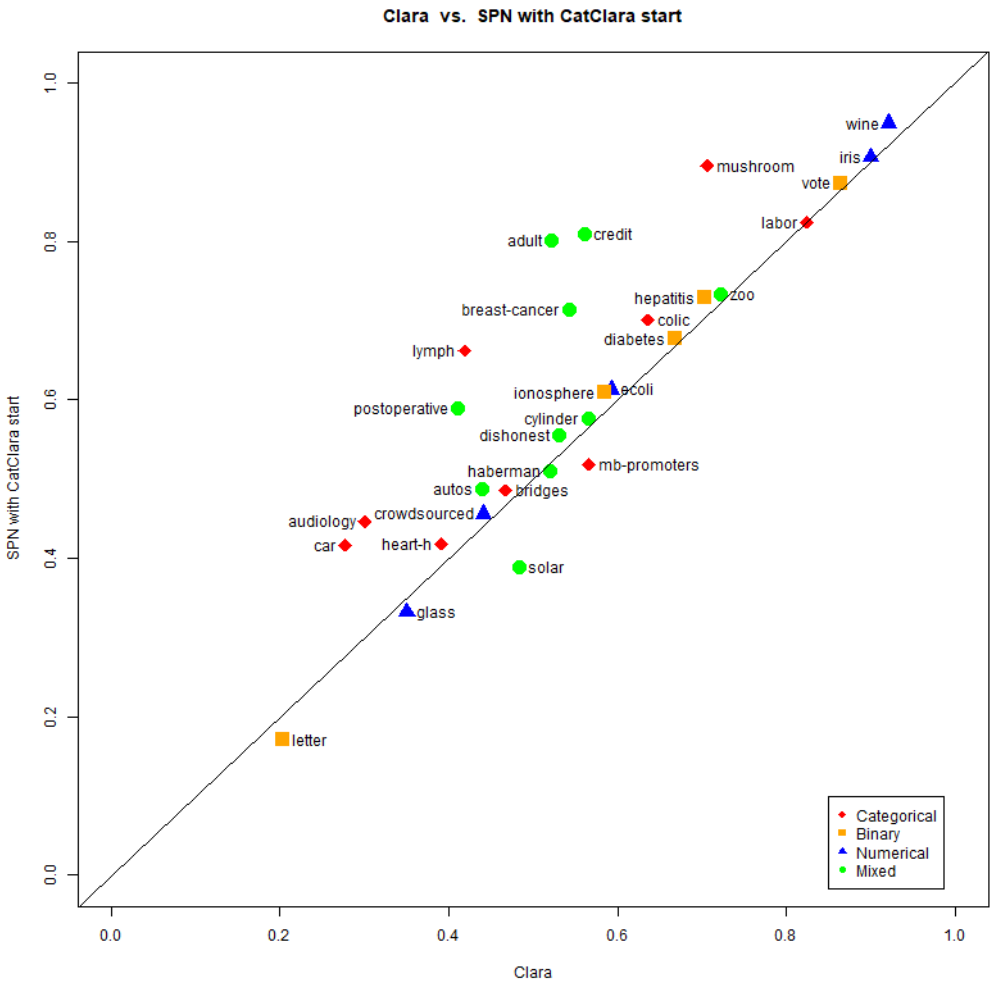

FIGURE 5.3: Results of SPN with a CATCLARA start plotted against the results of CLARA, averaged over 10 runs.

datasets. Compared to CLARA the SPN performed worse on Solar (-10%), MB promoters (-5%), Letter (-3%), Glass (-2%) and Haberman (-1%). On the other side, some of the datasets the SPN performed better on are Adult (+28%), Credit approval and Lymph (both +25%), Mushroom (+19%), Postoperative (+17%), Audiology and Car (both +14%) and Breast cancer (+12%).

5.4 Conclusion

When looking at these results, it becomes clear that applying the Sum-Product Network to the data results in a better clustering result in almost all cases. This improvement is especially evident for the experiment with a random starting distribution. Here, for all four types of datasets the SPN was able to deduce better clusters, even when the initial distribution held no informational value for the cluster. For the experiments where the initial distribution did contain some clustering information (by means of CATCLARA), applying the SPN also resulted in a higher accuracy. But for these results were only noticeable for categorical and mixed datasets. Nonetheless, when taking into account the heterogeneous datasets, it is very obvious that the Sum-Product Network with a CATCLARA starting distribution outperforms CLARA. Even

though the results of CLARA and CATCLARA were only slightly in favor of the latter, the results after applying the SPN for clustering were almost all improved compared to CLARA. In the cases where the SPN did not improve, it was a difference of only some percentages.

For the different values for α, the results were not all that different. For the SPN with a random start, none of the datasets resulted in a significant difference between the three values. This is mainly due to the fact that the standard deviations are rather high for some datasets. For the SPN with a CATCLARA start, the differences varied a lot. But because the model turned out to be deterministic for this specific experimental setup, the standard deviation of the results is zero for all datasets and α's. This means that all results that are not exactly equal are thus significantly different from each other. When looking at the outcome of the experiment, it becomes clear that a value for α of 0.1% has the best results. A smaller α means that less dependencies are found in the data, which leads to smaller SPNs. This, on its turn, means that small values for α are also faster than large values. To conclude, 0.1% is the best parameter to use for the model, both looking at accuracy and speed.

Notable is that the SPN with a CATCLARA start does not always show better results compared to CLARA on datasets containing categorical variables. After further inspection, it turned out that not all datasets labeled as such realy are mixed or categorical. For example, Zoo and Haberman are mixed datasets, but only of numerical and binary variables. These are the types which are not influenced by the changes made by CATCLARA.

Another possible explanation is that the model's definition of categorical variables does not correspond with our assumptions about them. We assumed that something is categorical when the distance between possible values has no meaning (e.g. DayOfWeek). But if we for example take the dataset Zoo, there is a numerical variable called Legs that states the number of legs the animal has. Fish have zero, birds two, most mammals four and insects six. Because the distance between the number of legs has meaning, this is not interpreted as a categorical variable. But if that is the case, the distance function will conclude that insects are more likelier to be clustered with mammals than with fish or birds, even though they all lay eggs. In this case it would probably had been better to interpreted as a categorical variable.

The same also happens vice versa. In dataset Solar, the intensity of the solar flares is described by a categorical variable with the values Low, Medium and High. In this case, the distance between Low and High should be larger than the distance between Medium and High. This variable should probably have been interpreted as a numerical variable.

6 Case study: Booking behavior

Now that we have established that the model is functional, we will do a case study. For this case study we will apply the model to various datasets with booking data to determine if it is possible for our model to be used to recognise and cluster similar booking behavior. We will try to define what the optimal number of clusters is and what the most important variables for the clustering are. On top of that, we will analyse the resulting clusters for recurring patterns and anomalies.

6.1 Data

The data that we will use for our experiments is taken from the booking process of real-life flights and was supplied by the airline for which this internship was done. The data concerns flights from Schiphol to certain destinations in Spain and vice versa over the period of January 2018 up to December 2018.

The data is divided into three data sets, namely *Madrid2018*, *Barcelona2018* and *SpainRest2018*. The first two data sets cover all flights to and from that specific city. The third data set is a combination of flights to and from several smaller destinations: Málaga, Ibiza, Valencia, Alicante and Bilbao. Some of these flights only operate in the summer period, so this data set is quite different than the other two.

Name	Instances	Variables	Numeric		Categorical		Classes
			Real	Int	Binary	Cat.	
Madrid2018	3,533	525	454	58	9	4	?
Barcelona2018	3,639	525	454	58	9	4	?
RestSpain2018	4,153	525	454	58	5	8	?

TABLE 6.1: The datasets with booking data used for the case study.

The data consists of rows of unique flights. A flight contains variables that determine its profile and features that describe the booking curve of that flights.

6.1.1 Flight profile

The first part of the data contains the specific variables that characterize the unique flights. This is information that has nothing to do with the booking data itself, but definitely influence the booking behavior (for example, some destinations and some dates are more popular and will thus have a higher demand). This flight profile contains the following variables.

Flight A combination of the flight number and the date on which the flight is scheduled to depart. Flight numbers are unique given a certain day, so this combination results in a unique identifier for each flight.

Carrier The carrier is the airline that operates the flight.

Date The date on which the flight is scheduled to depart. This date is mostly used to derive other features.

Month The month in which the flight is scheduled to depart. This is an important variable because airlines use different strategies for different months.

Season An indicator of which strategy is used depending on the period of the year. In the summer there is a large focus on tourism, while in the winter there is more a focus on business. The months of July and August are the so called summer peak. Here a very strict strategy is maintained, due to the fact that demand is very high. The remainder of summer is called summer shoulders. This period more or less coincides with the daylight saving time period

Special date A boolean that signifies whether there was a special event on that date. Special events are Christmas, Easter and any other national holidays in the country of origin and destination.

Day of Week The day of the week on which the flight is scheduled to depart. Within a week, overall there is a clear difference between days. Some for example have more business motive passengers than others.

Weekend A boolean that signifies whether the day of week is a Saturday or Sunday.

Departure Time The local time at which a flight is scheduled to depart from its origin.

Arrival Time The local time at which a flight is scheduled to arrive at its destination.

Bank The period of time in which the local departure time falls. Every day is divided into 8 banks of increasing length.

Bank 1:	06:30 - 07:45	(1:15)	Bank 4:	13:30 - 16:15	(2:45)
Bank #:	07:45 - 09:15	(1:30)	Bank 5:	16:15 - 19:15	(3:00)
Bank 2:	09:15 - 11:15	(2:00)	Bank 6:	19:15 - 22:30	(3:15)
Bank 3:	11:15 - 13:30	(2:15)	Bank 7:	22:30 - 06:30	(8:00)

Heading Whether a flight is inbound or outbound, as seen from the main hub of the airline. In our case, this main hub is AMS (Schiphol Airport).

Seasonality A combination of the month and the heading of the flight. This feature comes from the assumption that some month are each others opposite regarding inbound and outbound flights, e.g. people going on and returning from their holidays.

Direction The direction in which aircraft flies. This is comparable with the Heading, but this is an absolute value that does not depend on the main hub. The possible directions are East and West. This is especially useful when dealing with transatlantic flights or flights to the Far East.

Flight KM The number of kilometers of that flight.

Origin (or Destination) Region The region of origin (or destination) of the flight. The regions mostly coincide with the continents.

Origin (or Destination) Country The country of origin (or destination) of the flight.

Origin (or Destination) City The city of origin (or destination) of the flight.

Timeframe	28	27	26	25	24	23	22
DBDs	360-315	314-252	251-210	209-168	167-147	146-126	125-105

Timeframe	21	20	19	18	17	16	15
DBDs	104-84	83-70	69-56	55-42	41-35	34-28	27-21

Timeframe	14	13	12	11	10	9	8
DBDs	20-17	16-14	13-12	11-10	9	8	7

Timeframe	7	6	5	4	3	2	1
DBDs	6	5	4	3	2	1	0

TABLE 6.2: The days before departure for each timeframe.

Origin (or Destination) Airport The airport of origin (or destination) of the flight. This often coincides with the city of origin (or destination), except in the cases where cities have multiple airports.

Origin (or Destination) Latitude and Longitude The coordinates of the airport of origin (or destination). This variable does not come with the rest of the data, but was extracted from 'openflights'. This website contains, amongst others, the exact location and coordinates of every airport in the world[1].

Aircraft The type of aircraft that is used for the flight. This is the main factor that determines the maximum number of seats available.

6.1.2 Booking curve

The second part of the data comes from the booking curves. As described in section 2.3, these curves contain the number of bookings for every day before departure and plotting them thus gives the overall booking behavior of the flights. This booking curve contains a lot of information for all these 360 days before departure. To summarize this information a bit, this period was divided into 28 timeframes. The first timeframes, the furthest from departure, extent over multiple weeks, while the last timeframes only have a span of a single day, as can be seen in table 6.2. This is because fluctuations in the final days of the booking process have more importance and a larger impact than for example fluctuations in the first hundred days.

For each of these timeframes the behavior of that section of the booking curve is described using some basic variables, namely

BLF The percentage of bookings relative to the capacity on the last day of the timeframe.

ΔBLF The change in percentage of BLF between the first and the last day of the timeframe. In the general case this variable should be a positive number.

Max ΔBLF Within a timeframe the BLF can fluctuate, resulting in local maxima and minima, possibly higher or lower than the first or last BLF of the timeframe. This feature takes those fluctuations into account and describes the largest difference of BLF within a timeframe.

[1] https://openflights.org and https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat, retrieved 23-11-2018

ELF The predicted BLF at DBD 0, based on the load factor on the last day of the timeframe compared to that of 5 related flights of the previous year.

Capacity The number of available seats on the last day of the timeframe.

ΔCapacity The change in percentage of the capacity, between the first and last day of the timeframe. Usually the capacity is fixed, but sometimes the aircraft is swapped for one of another type, for example for operational or maintenance purposes. Such drastic change in capacity also drastically impact the BLF as this is relative to the capacity.

The BLF is the basis of the booking curve, but there are multiple ways the BLF can be split up to further specify the behavior of the bookings. Amongst others, it can be split into cabin, traffic type, travel motive and group bookings. These variables have been selected by analysts as being the most predictive.

When such a variable states that a certain percentage of the BLF belongs to this type, this actually means a percentage of the capacity. To show, assume the capacity of a flight is 100, the BLF is 90% and it is said that 80% of the BLF belongs to a certain type. In that case there are 80 bookings that belong to that type (80% of 100 seats) and not 72 (80% of 90% BLF).

6.1.2.1 Cabin

The clearest distinction between bookings is the cabin in which they are booked, namely economy or business. Which cabin a booking is in depends on its booked subclass. For both cabins the capacity is known, so the load factor of a cabin is relative to that capacity. Therefore it is possible for all cabins to be at 100% load factor.

This resulted in the following data.

Business BLF The percentage of business cabin bookings relative to the business cabin capacity on the last day of the timeframe.

Business ΔBLF The change in percentage of business BLF between the first and the last day of the timeframe. In the general case this variable should be a positive number.

Economy BLF The percentage of economy cabin bookings relative to the economy cabin capacity on the last day of the timeframe.

Economy ΔBLF The change in percentage of economy BLF between the first and the last day of the timeframe. In the general case this variable should be a positive number.

6.1.2.2 Traffic type

On every flight there are customers with a lot of different origins and destinations. These are not just the origin and destination of the flight itself, but also those of connecting flights before or after. Some flights are more suitable than others for connecting to other flights, for example because of their arrival time at the destination.

Considering these connecting flights, there are four different categories. There are customers that just fly from the origin to the destination and nothing more. This is the Local traffic type. Customers that have a connecting flight only to or from somewhere in Europe (i.e. Medium haul) belong to the MH-MH traffic type. Customers that fly from somewhere in Europe with a connecting flight to an intercontinental destination

(i.e. Long haul), or vise-versa, belong to MH-LH traffic type. Finally, there are customers that depart from an intercontinental destination and have a connecting flight to another intercontinental destination. This is the LH-LH traffic type. All traffic types but Local, require at least one connecting flight, but it does not matter how many.

This resulted in the following data.

Local BLF The percentage of the BLF with local traffic.

MH-MH BLF The percentage of the BLF with HM-HM traffic.

MH-LH BLF The percentage of the BLF with HM-LH traffic.

LH-LH BLF The percentage of the BLF with LH-LH traffic.

6.1.2.3 Travel motive

Even though the travel motive looks very similar to the cabins, there is an important difference. Cabin is a very clear distinction based on subclass, while travel motive is an calculation based on the length of stay. The length of stay is the number of days between the departure and arrival date of two consecutive flights in a booking. If there are multiple flights, the longest length of stay is used. In the case of a single flight, the length of stay is unknown. In this case the travel motive is One way. When the length of stay is 2 days or less and none of these days is in the weekend, the travel motive is Business. One way motive falls under the same price category as Business motive, so they are grouped together. All other bookings are considered Leisure motive.

This resulted in the following data.

Leisure BLF The percentage of the BLF with leisure travel motive.

Business/One way BLF The percentage of the BLF with business or one way motive.

6.1.2.4 Groups

As stated in section 2.3, group bookings can be a problem in defining the booking behavior of a flight. Group bookings are special bookings for groups of at least 10 people. They are often made by travel agencies beforehand and are no direct indication of customer demand, nor their willingness to pay. These agencies have a certain time limit before which they have the rights to alter or cancel their bookings for a minimal fee. If this time limit passes the normal cancellation fees are from now on applied to the booking. As of then, the group booking can be handled just as normal bookings and are thus an indication of customer demand. Group bookings that are not yet certain can cause a lot of noise in the curve. Because it always concerns large amounts, a single group booking has a large impact on the BLF.

To prevent the early group bookings from misguiding the clustering algorithm to much, this resulted in the following variables.

Group BLF The percentage of the BLF being part of a group booking.

Group ΔBLF The change in percentage of group bookings BLF between the first and the last day of the timeframe. Preferably, this would be a positive number, but due to minimal cancellation fees, especially in the early timeframes, this will fluctuate a lot.

6.2 Methodology

To determine if it is possible for our model to be used to recognise and cluster flights with similar booking behavior, we will run multiple experiments with our model on the various datasets with booking data. Part of these experiments will be to determine the optimal number of clusters for each dataset. To determine this number, we will run the model on each dataset with three different number of clusters, namely 15, 30 and 50. We will also try to find what variables or characteristics are the most importance for describing the booking behavior. The approaches of how we will determine what these optimal results are and how we can validate them, will be further described in section 6.2.1.

For all experiments in this case study, the SPN will be run with a CATCLARA start and with an α of 0.1%, as the effectiveness of these parameters was concluded from the testing experiments in chapter 5.

6.2.1 Validation

For this case study, we cannot use the same way of validating the results as for the the testing experiments. For those standard datasets the correct class labels were known and could therefore be used to calculate the real accuracy of the model. Unfortunately, there are no class labels available for the booking data. Because of that, we need to find another way of validating the quality of the results.

The most effective way is to discuss the results with analysts, them being the domain experts. The difficulty is that for this we need a structured way of displaying the results, because it is impossible to discuss the cluster assignment of every single flight for datasets of over 3,500 flights. Therefore we need to give a summary of every cluster that describes the flight profiles and the booking behavior that are contained in it. Plotting the booking curves of all flights that are clustered together should give a good indication whether they display similar booking behavior. The booking behavior can then also be easily summarized by taking the averaged booking curve of all flights. Besides the visual representation we will also generate a list of profile characteristics of each cluster. These characteristics are not used for clustering, so they will not influence the results.

For this summary of the clusters, we will split the data into several sections, corresponding to the distinctions described in section 6.1.2. As a result of splitting the data this way, we can also determine what variables contain useful information to describe the booking behavior.

To determine what an optimal number of clusters is for the data, we will look at the number of flights in each of the generated clusters, that is, the size of the clusters. Depending on the distribution of the sizes we will try to infer at what number of clusters the results are in the elbow of the graph, as described in section 3.4.2.

6.3 Results

Some of the results of the case study are shown in Appendix B. In these figures the booked load factors are plotted against the days before departure, as we saw in section 2.3. Sections B.1, B.2 and B.3 show the average booking curves of the 30 resulting clusters, when clustered on flight profile and booked load factor. In sections B.4 and B.5, different booking behaviors are displayed. Finally, in section B.6 different distributions of traffic type are illustrated from clusters containing specific banks. The exact lists of profile characteristics for all these clusters are not included because of

their confidentiality for the company. Despite this fact, these characteristics are still described and used for some cases.

The first experiment was to cluster the flights on their profile. When looking at the results, the first observation to mention is that the flights are distributed fairly equal over the clusters. Most clusters contain between 100 or 200 flights, with some smaller clusters containing around 50 flights. Next to that, a few cluster only contain a single flight. The characteristics of these profile clusters show that within the clusters there generally is the least variation between the season and between the direction. This means that most of the flights within the each cluster have the same season and the same direction. Secondly, their bank is in a number of cases also fairly similar. Even though they usually are not limited to a single bank, the different banks are often adjacent to each other. Some very small clusters also show a cohesion around a certain day of week (e.g. weekend), but this relation is lost in the large clusters.

When looking at the average curves of the clusters, they all show similar shapes. There are individual outliers, but in general the curves all gradually grow towards the 80% to 100% at 0 days before departure. The curves do diverge a bit around 150 days before departure, but other than that most curves are not really unique.

On the contrary, when looking at the results that were clustered on the booked load factor, almost every average curve has its own distinct pattern. Many curves have a lot of fluctuations and are less gradual. The gradual curves that are comparable to the curves of the profile clusters, however, are unique, for their curve or final loadfactor all differ. On the other hand, the common characteristics of the clusters are much more mixed up. Especially the bigger clusters do not have specific characteristics. The season, direction and bank are not nearly as distinguishable. However, some very small clusters of only a couple of flights do have specific characteristics. These clusters were able to group flights on the same special dates. For example, 4 flights in a cluster of 5 departed during Easter and in a different cluster of 3, all flights were on All Saints' Day, a national holiday in Spain.

6.3.1 Cluster size

From the results in table 6.3, it becomes clear that there is also a larger variation amongst the number of flights in each cluster. Most flights are grouped together into a couple of very large clusters of hundreds of flights, up to a thousand. After that there are some smaller clusters, ranging from a hundred to a few dozen. Finally, there are quite some clusters that only contain a few flights or even only a single one.

	Nr of clusters			Nr of clusters			Nr of clusters		
Size	15	30	50	15	30	50	15	30	50
250+	6 1	5 1	5	5	5	5 1	5 1	5 1	5
100-249	2	4	5 1	4	4	5	5 1	5 2	5 4
50-99	2	1	5		3	2	1	5	5
25-49	1	2	2		4	5	1	2	5 2
5-24	2	5 2	5 5 5	5	5 4	5 5 1		4	5 5 5 1
1-4	1	5 5	5 5 5 2	1	5	5 5 5 1		5 1	5 3
	Madrid2018			*Barcelona2018*			*SpainRest2018*		

TABLE 6.3: The distribution of the cluster sizes for the three datasets clustered on booked load factor.

When using 30 clusters, for both *Madrid2018* and *Barcelona2018* more than half of the clusters is smaller than 25. For *SpainRest2018* this is a third. For the clusters larger than 100 flights, these numbers are reversed, being a third for the first two and almost half for the latter. When the number of clusters is reduced to 15, the number of large clusters stays approximately equal. But on the other side, the number of small clusters is reduced drastically. In the case of *SpainRest2018* even to none. For increasing the number to 50 clusters, quite the opposite happens. There are slightly more large clusters, but most of the new clusters are very small in size. These small cluster make up almost 60% of the total number of clusters, while only containing 5% to 10% of the total flights. On the contrary, the clusters larger than 250 flights make up only 5% of the total number of clusters, but at least 60% of the total flights.

The averages of each cluster all have their own distinct pattern. But regarding the individual flights within each cluster, though they all have the same general shape, there still is enough diversity amongst them. Especially considering the large clusters, where in the first 300 days before departure this diversity is very clear. In this period for some flights, high plateaus rise up 20% to 40% above the other curves, often only to drop again a couple of days later. These plateaus continue to arise until departure, but they do get narrower the closer they get to departure, resulting in sharp peaks protruding from the curves. In the last 60 days, the curves generally converge towards a distinct shape. Still, the final booked load factor can differ even up to 40% within a cluster. As examples of this, see figures B.7 and B.9.

The flight in the smaller clusters often have a more unique shape than those in the larger clusters. In most cases they also do have more fluctuations around a clear centroid. Even when they have a similar shape to those of a larger cluster, there usually is a distinct feature that sets them apart. See figure B.10 for an average shaped curve, but with the distinction of a major drop of almost 20% around 21 days before departure. Other distinctions can be that all flights have a plateau or a very sharp increase at a certain period. An example of this is figure B.12 where around 25 days before departure all flights have a sudden and sharp increase. But even when excluding these distinct features, the curves still have the same general shape.

6.3.2 Traffic type

From the flights that were clustered on traffic type a distinction that became clear is that of the relation with the bank in which a flight departs, especially for *Madrid2018*. When looking at characteristics as well as the plotted curves, there were multiple clusters for which approximately 80% of the flights were in the same bank, even though the day of week was very mixed. The differences in traffic are mostly between Local, MH-MH and MH-LH traffic. LH-LH traffic is negligible for all flights in the discussed clusters.

Figure B.13 shows the distribution of traffic type for the cluster that belongs to bank 2. Here, the MH-MH and MH-LH traffic start around 275 days before departure and both end up with more than a third of the final capacity. The local traffic began booking much later and accounts for less than a quarter.

For the cluster that contained mostly bank 4 and 5, bookings for local traffic come in 100 days later, but they make up for more than half of the flight, as can be seen in figure B.14. MH-LH traffic gradually has some bookings, but is surpassed by MH-MH traffic at 10 days before departure. At departure, they both fall just short of 20% of the final loadfactor.

In figure B.15, which describes the flights departing in bank 6, almost no bookings arrived before 150 days before departure. At first, the MH-LH traffic starts booking,

but is exceeded by local and MH-MH traffic, at 60 and 45 days before departure respectively. Of the final loadfactor, a third consists of local traffic, followed by MH-MH traffic 5% below it.

For bank 7, Figure B.16 shows that almost all passengers are local traffic, with the others having a total share of only 10%. Bookings of local traffic already start coming in as early as 250 days before departure, while the others start only after 150 days have passed.

6.4 Conclusion

After we collected these results, we evaluated them with the domain expert. During this meeting, we discussed the quality of the results (section 6.4.1), their possible impact on the daily way of working (section 6.4.2) and the options for future improvements (section 6.4.3). We will also draw conclusions from this case study. This conclusion and discussion are mostly concerned with the business perspective of these experiments and less with the scientific aspects.

6.4.1 Evaluation

After inspecting the results, the analyst was positive about the outcome. The large clusters, though looking very tangled at times, clearly display flights with similar booking curves. The combined booking curves make it easy to see that there are apparently a few common shapes the booking curve of a mainstream flight can have. The large clusters each show one of those shapes. The curves in figure B.8 display late booking behavior, while the curves in figure B.7 display very early booking behavior. However, having a similar curve is not the only factor determining whether flights have similar behavior. Therefore, a downside of the current results is that the variation in booked load factor at the final day before departure can be quite large. For determining the exact behavior of a flight, this last period is very important. In the case of figure B.7, the variation is too large, while for figure B.9 it is acceptable. But for these figures it is vice-versa when taking into account the variance in curvature leading up to the departure.

On the other side, considering the smaller clusters, as we also saw in the results, the model was able to cluster flights that flew on or during the same holiday or event. The data that was used for these clustering results did not contain any variable with a direct link to the event (no date or flag). The model was able to group those flights solely on their booking behavior.

An example of this is a big yearly event in Barcelona. In 2018, Mobile World Congress (MWC) took place from February 26th to March 1st. As a result, on the days before the event all flights to Barcelona were booked by a lot of business traffic. And after the event, the flights to Amsterdam were booked by the same, returning business passengers. According to the analyst, if there was anything the model was suppose to detect as deviating, it should be the flights around this event. And it did. We analyzed the results that were clustered on cabin load factor and noticed that 36 of these flights were clustered together, in a cluster of 52 flights (see figure B.11). These flights all have a very late booking behavior because of the business traffic and an almost completely full business class. This was what the model recognised, since the only profile characteristic that these flights have in common is their time period (i.e. their date), as their bank, day of week and direction all differ.

Another example is a cluster where the model put together flights with similar behavior due to operational changes. Figure B.12 shows a cluster of flights where

around 25 days before departure the aircraft was changed to a smaller type. Such an operational change is sometimes made when a flight is not expected to reach a sufficient load factor. A smaller aircraft is then deployed as a cheaper alternative without causing any hindrance to the passengers. As a result, the total capacity of those flights decreases while the number of bookings stay the same. Because the booked loadfactor is the number of bookings relative to the total capacity, it increases by around 20% without any new bookings. The model clustered these flights based on the sudden increase of booked loadfactor, but also on a high ΔCapacity. Therefore, even though the curves look very similar to those in figure B.11 (flights to MWC), they certainly have different booking behaviors.

Also, prior to the experiments, the analyst expected that when simply clustering on booked load factor, group bookings would cause lots of noise. She predicted that flights with a lot of early group bookings would be clustered together, regardless of the rest of the curve. These group bookings are the plateaus we identified while describing the results. Because flights with those group bookings are clustered into the mainstream clusters, this signifies that the model was able to ignore the noise of group bookings. Despite the prediction of the analyst, the model proves it can handle group bookings. Admittedly, there are some smaller (<25) clusters where group bookings are the distinct characteristic, but this is the exception rather than the rule.

Finally, for the flights that were clustered on traffic type, the results are very clear. The clusters of which the majority of flights departed during a certain bank, show a very evident distinction. Madrid, being a well known transfer hub, has a lot of non-local traffic. The results evidently show that this traffic mostly travels on morning flights and also a bit on evening flights, while the afternoon and night flights are more used by the local traffic to travel between Amsterdam and Madrid only. This is a known fact for analysts, because these morning and evening flights are much more suitable for connecting to other flights. But despite the fact that these results were already known, it still shows that the model can detect dependencies like this in the data.

6.4.1.1 Cluster size

The pattern that occurs when changing the number of clusters seems to be very obvious. When more clusters are added, only small groups of flights get put into them, while the majority of flights stays in roughly the same few clusters. As evaluated, the flights in these small clusters appear to be deviating flights, while the flights in larger clusters are more mainstream. The horizontal line in figure 6.3 between cluster size 49 and 50 is the separation between these two types of clusters. According to the analyst, which type of cluster is more useful depends on the desired application of the model and its results.

Changing the number of clusters will change the distribution of the cluster sizes. If the number of clusters is lower, the clusters will generally be larger. And when the number of clusters is higher, there will be a lot of small clusters. The only exception here is *SpainRest2018*. Here also the number of larger clusters increased when the number of clusters was set to 50. This is caused by very large (250+) clusters splitting up into 'smaller' (100+) clusters. Our assumption about this exception is that this dataset is compounded of multiple destinations. Therefore it is possible that flights to those cities differ a bit between themselves, making a split in the large clusters more plausible.

6.4.2 Application

According to the analyst, the results that were presented (and thus the model) can be applied to impact the current way of working. There are two main applications for which the model can be used. The first being on a large scale with less focus on individual flights and more on the overarching strategies, while the second is on a much smaller scale and focuses on the behavior of single flights.

For the first application the clusters can used in the process of determining steering strategies. Every so often, for every geographical complex the overall performance is analyzed and assessed. If needed, the steering strategies will be adapted. Having the clusters at hand can give extra information for example about how a certain daily flight behaves differently on different days or in different months. Especially when looking at more than one destination, this can give insight into similarities that can lead to aligning steering strategies. For this, the large clusters give the most relevant information because they encompass the more common booking behavior, since the others mainly contain deviations. Or to put it in the words of the analyst, for this purpose clusters smaller than 50 are not even worth looking at.

The second application can be demonstrated by an example. Take the cluster containing flights to and from MWC in Barcelona, as discussed earlier. In this cluster two-third of the flights have a clear explanation of why they are clustered together, being the same event. This leaves one-third of the flights that were not during MWC but were put in this cluster based on their cabin loadfactors. It is therefore interesting to find out why those flights have similar behavior. Out of all the thousands of flights, this provides specific flights for the analyst to conduct a closer investigation. This might lead to better or new insights into the behavior of those specific flights or dates that on their turn might lead to an improvement of future strategies or steering. This can be done for every cluster with distinct characteristics but without a clear connection between the flights. Clusters without distinct characteristics can still be used to detect historical outliers, but this research is less directed. Because the large clusters contain most mainstream flights, those flights are less interesting for investigation than the (deviating) flights in the small clusters. And that is a good thing, because at the current stage in the model's development, this investigation has to be done manually, and will therefore only be feasible for relatively small clusters.

6.4.3 Future improvements

Even though it already has possible applications, there are a number of ways the model can be further improved to increase its performance and reliability. Some of these have to do with the data or the way we interpret it, while others are more concerned with new implementations.

This first improvement, we already brought in during the evaluation of the results. Within the large clusters, the variance in booked load factor in the final days before departure was to high. To prevent this, the data or the model could be adapted to grant more weight to this last time period. The current approach of using fixed timeframes was a genuine step in the right direction by granting the last 10 days the same weight as the first 300 days (see table 6.2), but this is not sufficient. Giving more weight to these variables can be done explicitly or during the normalization. By normalizing these variables to values between 0 and 10 in stead of 0 and 1, a minor difference already is counted as ten times heavier in the calculation of the distance than the other variables. Another possibility is adding more variables to closer describe the behavior of the bookings in this final time period. To the variables described in section 6.1.2, new variables could be added that take into account the ΔBLF over multiple, adjacent

timeframes. This way the model should be able to reduce the variance in booked load factor in the final days before departure.

A second possible improvement also deals with an introduction of new data. Now that the model can cluster flights based on the shape of the booking curve, the next step would be to describe the booking behavior even more accurate. If two flights have identical booking curves, this does not always guarantee that they also have identical behavior. The process of steering flights means that an analyst is constantly reacting on new developments in the incoming bookings by adapting the subclass and other price settings. This influences the new bookings, on which the analyst has to react again. If therefore two flights have identical curves, but one required almost no steering because it gradually followed the desired behavior, while the other demanded constant effort to steer it towards this behavior, there booking behavior is clearly different. For the model to know this extra layer of booking information, new variables should be selected that give an indication of the amount of steering effort that had to be put in. These could be indicators of potential spoilage or spillage or of the minimal ticket price on each day before departure (i.e. bidprice).

Finally, the analyst suggested another possible implementation of the model. In the current situation the model is backward looking. This means that it only handles flights that have already flown. This can be very useful for analysis and deepening the analyst's understanding, but cannot directly support the analyst with the actual decision making on flights that are still in the booking process. For this, the model should be made forward looking. This can be done by iteratively learning the model with backward looking data, as was done in our experiments, but in stead of using the resulting clusters, the resulting SPN is stored. This SPN can than be used to assign flights that are still in the booking process to any of the clusters. Because of the nature of SPNs, the variables that are not yet available can be omitted and it will still return the cluster with the highest probability for that flight. This gives information about the current status of the flight, but even more so when done daily. When a flight is then assigned to a different cluster than the day before, the analyst could receive an alert. This way the model can take over a part of the daily work of analysts and leaves them more time to spend on other important projects or developments.

Continuing with this idea of making the model forward looking, another future application becomes possible. On top of clustering the provisional curve and recognizing its behavior, the model could also be used to predict the final part of the curve. Using the inference process described in section 3.2.2 for the situation where not all variables are known, the model should return the most probable values considering the curve so far and its assigned cluster. This would even further support the analysts with their decision making.

6.4.4 Discussion

On the whole, the analyst was content with the results. The general booking curve shapes of the large clusters and the distinct characteristics of smaller clusters show that the model is able to cluster these flights into logical and meaningful manner. Clusters like the flights to Mobile World Congress that the analyst expected, were present in the results, while the predicted noise of clusters based on group bookings stayed out. However, there are still a number of improvements possible to further increase the quality and usability of the results.

We have also established that some variables or approaches are essential for specifying booking behavior. Cabin can be used to detect special, high-profit events and traffic type can distinguish between different banks. Capacity is necessary to identify

sudden increases in the load factor as operational changes. Splitting the days before departure into timeframes turned out to be a valid way of canceling out the potential noise caused by group bookings.

Likewise, we came to the conclusion that the optimal number of clusters for a data set depends on the desired application of the results. If the application is to investigate the behavior of individual flights, 50 clusters is the optimal number that was tested in these experiments, because this number resulted in many small clusters that allow for deeper research. Otherwise, if the application is to be used for determining steering strategies, 15 clusters is more optimal. This is because almost all flights are then contained in a few large clusters, making it suitable for a general analysis.

There are two more thoughts concerning the optimal number of clusters. Firstly, a suggested method of determining this optimal number was by calculating the variance in the booked load factor at certain DBDs for the large (100+) cluster. The elbow can than be determined as the point where the variance decreases less. The DBDs that were to be selected are mostly close to the departure day, because there the variance should be as low as possible (as discussed before). This assumes that the smaller clusters mostly contain outliers, so by putting them in separate clusters, the large clusters should only contain flights with more ordinary booking behavior with similar final booked load factor. The problem with this method is that the model was not build to minimize the variance at the last days before departure, but to detect similar behavior over the whole length of the curve. This approach is therefore not guaranteed to result in the characteristic elbow-shaped curve. However, if a future improvement is made that specifically focuses on these final days, this would be a viable method of determining the optimal number of clusters.

Secondly, the trend between 15, 30 and 50 clusters seems clear, but it would be interesting to see how it continues for huge numbers, say 200. Most of the time the number of clusters is increased, it creates an new small cluster. The number of large clusters stays the same, but these clusters do slowly get smaller as more and more outliers are placed into separate clusters. Do the large clusters at some point disappear or will the small clusters be split up first? And if for a huge number of clusters there are still some large (100+) clusters, what can we conclude from that? Will this than be an even more ideal number of clusters, as all outliers have been removed from those large clusters, making them perfect for general analysis, and as the many small clusters with outliers allow more than enough for specific investigation? Or will it just in a huge amount of small clusters without specific characteristics? Because of the potential results, this is an interesting subject for further research.

7 Conclusion

In this thesis, we have researched whether it is possible to use a Sum-Product Network for clustering. For this we implemented a clustering algorithm that iteratively learns Sum-Product Networks and uses them for inference. On top of that, we implemented our own version of CLARA that can correctly handle categorical variables: CATCLARA. To test if the model is functional, we ran the algorithm on various standard datasets. We also tested different parameter settings, namely three different starting distributions and three values for α. Secondly, we performed a case study on booking data of flights, to test if our model can cluster such data in a meaningful way. For this we divided the data into timeframes and selected multiple features to describe the behavior of the bookings during that timeframe. We likewise experimented with different number of clusters to determine what the optimal number to use is.

Now, we answer the sub research questions that were presented in section 3.5. After that, we will answer our main research questions stated in section 1.2 and discuss choices we made during the research and possible options for future continuation.

1. *What adaptations or implementations are required for a Sum-Product Network to be used for clustering?*

   The main implementations that were needed for the SPN to be used for clustering are implementing the Hard Expectation Maximization steps and defining a correct distance function for the clustering. On top of that we made some adaptations to the SPN to optimize the performance and the quality of the results.

   The distance function that we defined is an adaptation of CLARA that can handle numerical as well as categorical variables: CATCLARA. This function splits the categorical values into separate columns. Each column has adjusted values so that they sum (or square) to 1 when two instances have different values, using Manhattan (or Euclidean, respectively), without any prior analysis of the data.

   For the Expectation Maximization steps, we implemented the learning of the SPN (as presented by Conaty et al. (2018)) in the E-step, including CATCLARA for clustering. We adapted the learning of the SPN to make it able to handle integer as well as continuous values. The inference on the SPN was implemented in the M-step, to update the cluster labels for all instances of the data. In the convergence step, we used an implementation of the Hungarian algorithm to compare the cluster assignments of the current and previous EM-step. This assignment algorithm makes it possible to work with very large numbers of clusters, compared to the implementation of convergence using permutations.

   To improve on the computational time for inference, we adapted the SPN so that the children of each node are ordered on size. During inference, the value for each product-node is iterative calculated from the smallest child to the biggest. Once a child returns 0 as value, the product node returns this value without calculating the value of its other children. Secondly, we adapted the SPN so the inference can be done concurrent on multiple cores. Another minor adaptation was to remove the max-height feature from the learning of the SPN, because this only worsened the results by forcing an early stop.

2. *What combination of parameters will lead to the optimal results in the experimental setup?*

   In the experiments, we compared five different clustering approaches. These clustering approaches were: Random, CLARA, CATCLARA, SPN with Random start and SPN with CATCLARA start. The first three approaches only generated an initial distribution of which the accuracy was calculated. The last two approaches used SPNs to improve upon the initial distribution until convergence.

   Out of the three approaches of generating the initial distribution, Random clearly performed the worst. When looking at the numerical and binary datasets, the results of the other two are identical, which makes sense because CATCLARA only changed the way it interprets categorical variables. When looking at the results for those datasets (categorical and mixed), CATCLARA is only slightly better than CLARA. For the two approaches applying SPNs, for most datasets, the SPN with CATCLARA start performed better than the SPN with Random start. Though, SPN with Random start did have better results on all datasets than Random. Finally, the comparison between CLARA and SPN with CATCLARA start is predominantly in favor of the latter.

   We also tested three different threshold values for the independence test used for leaning the SPN. The values for this α that we compared are 2%, 1% and 0.1%. While there were almost no significant differences for Random start, for CATCLARA start, 0.1% clearly had the best results. A smaller α means that less dependencies are found in the data, which leads to smaller SPNs.

   To summarize, the combination of an SPN with CATCLARA start with a threshold α of 0.1% led to most optimal results in the experimental setup.

3. *How can the quality of the resulting clusters be validated?*

   We validated the quality of the resulting clusters in two different ways. The datasets of the testing experiments had true class labels available, so we could calculate the actual accuracy of our predicted results compared to those. The difficulty with comparing cluster is that the assigned labels do not matter, but only which points are clustered together. Therefore, it is possible that the same cluster is present in both the true and predicted results but with different cluster labels. For this, we also used the Hungarian algorithm that is used for calculating convergence for the same reason, to determine the assignment (or permutation) of cluster labels with the highest accuracy. This accuracy is then used to represent the quality of the resulting clusters.

   The booking data had no true labels as benchmark, therefore another method of validating the resulting clusters was necessary. For this, we used the domain knowledge of an inventory analyst to assess the quality of the results. The clusters were presented in two representations. The first is a visual depiction of the booking curves. The graph could display the average booking curve of each cluster and the curves of all flights within a single cluster, as well as the distribution over the specific variables used (e.g. traffic type). The second representation is a list with profile characterizations of the flights in each cluster. These characterizations are variables and combinations of variables from the flight profile (e.g. bank, day of week, special date, seasonality), together with the number of occurrences. On top of that, we did an analysis on the number of flights in each of the resulting clusters, presenting the distribution of large and small clusters. Based on these representations, the analyst gave her opinion about the quality of the resulting clusters.

4. *Is it possible to detect flights with deviating behavior by looking at the clustering in combination with 'sibling' flights?*

   After examining the outcome of the case study, we can conclude that it is indeed possible to detect flights with deviating behavior when looking at the clustering. These deviating flights were mostly caught in the smaller clusters, while the more mainstream flights formed large clusters. By increasing the number of clusters, the number of small clusters that contain outliers also increases. These clusters can often be defined by a distinct feature in their booking curves, like a sudden drop or increase in booked load factor, or by a distinct profile characterization, like a specific bank or event.

   However, due to time constrains, we were not able to determine a clear definition of 'sibling' flights. The closest we came to this definition were the clusters that were clustered based on flight profile. When these clusters are compared to the clusters that were clustered based on booked load factor, a clear difference was visible. When looking at the averages of both clusters, the averages of the clusters based on flight profile all show similar shapes, while the averages of the clusters based on booked load factor all have their own distinct shape. If the first clusters are used to define 'sibling' flights, the comparison with the second clusters show that these clusters clearly assign the flights to different clusters. Further analysis should reveal into which clusters flights that have the same profile are clustered based on booked load factor. If some 'sibling' flights are clustered together, while some others are spread individually over other clusters, this can be a sign of deviation. Hence it is likely that 'sibling' flights can be used to detect flights with deviating behavior.

Is it possible to use a Sum-Product Network for clustering and analysing the booking behavior of flights?

To conclude, in this thesis we have demonstrated that it is indeed possible to use a Sum-Product Network for clustering and analysing the booking behavior of flights, in such a way that it satisfies both the business and the scientific motivation. By implementing the SPN as an Expectation Maximization algorithm and using optimal parameter settings, we were able to cluster the heterogeneous data of the flights in a meaningful way that offers enough opportunity for further analysis. Beforehand, we proved the effectiveness of this model by testing it on multiple standard datasets. The results of these tests show that the SPN with CATCLARA start, achieves a higher accuracy than the established clustering algorithm CLARA. For the case study, we evaluated the results with an analyst and they were acceptable, but she had multiple suggestions for further developments. The model was able to detect specific relations between variables and recognise flights with deviating behavior, but it did not improve much upon the domain knowledge of the analyst. However, it is a valuable first step towards a model that can more extensively support analysts in their decision making.

7.1 Discussion

Concerning the results, though they demonstrated that our SPN with CATCLARA start performed better than CLARA, this was not as much as we initially expected. Especially on some of the categorical and mixed datasets we were expecting a bigger difference in accuracy. On the other side, this is not all that surprising since CLARA is a respectable clustering algorithm in itself. In most cases, the SPN only need a couple of iterations to converge after the CATCLARA start, while with a random start it

would sometimes need as much as 30. Apparently CLARA and CATCLARA generate an initial distribution that is much closer to a local optimum with a higher accuracy. This gets confirmed by the fact that standard deviation for these experiments is always zero, meaning that with each run it always reaches the same local optimum. For the datasets where the SPN has a lower accuracy than CLARA, the initial distribution was probably already close to local optimum and by updating the classes, the SPN jumped to a lower optimum in the fitness landscape.

Nonetheless, using a version of CLARA in the SPN and subsequently comparing the outcome with that of CLARA, results in a fair comparison. Every improvement that is made is directly caused by implementing the SPN around CLARA and can therefore prove the effectiveness of applying SPNs for clustering.

While the scientific side of the experiments of implementing the model is very important, for the business determining what data to use is of equal importance. The variables that we used in the case study were decided upon together with the analyst because they are used in their daily work. The features were then selected to best represent the behavior of the curve during that timeframe. During the experiments, we tried to implement some simple feature selection using the structure of an SPN. Based on where a variable occurs in the network and where it splits of, we tried to extrapolate the importance of that variables. Unfortunately, the network of an SPN is very hard to interpret, because of the alternation of sum and product nodes. Therefore, this sidetrack did not yield any clear or meaningful results. Performing feature selection of some sort would have been a nice addition to the results, giving a better understanding of the usefulness of the selected variables for clustering flights.

7.2 Future work

For this thesis, we made a number of choices about the implementation of the SPN. Because it was our goal to determine if it was possible to use SPNs for clustering, this served as our main motivation to choose simple variants of the required implementations. Now that we have shown the effectiveness, there are multiple areas where further developments can be made. We will discuss some options for future work on clustering with SPNs.

One of the choices we made was implementing the Expectation Maximization using the hard approach. As explained before, with hard EM, each instance gets assigned to the cluster with the highest probability, without regarding if there are also other clusters with high probabilities. This makes the implementation a lot simpler, but this comes with the cost of losing potentially very valuable information, namely the membership of an instance to each cluster. Soft EM stores a list of probabilities for each instance, retaining this information. This information can be very useful when comparing cluster, because if multiple instance have high probabilities for both, they are probably similar. However, implementing soft EM will require quite some changes to the SPN and it also poses new challenges in terms of presenting and validating the results, since this already posed enough challenges for hard EM. Though, further research in this direction will most certainly be worth the effort.

Another choice we made was for one specific learning method of the SPN, based on the work of Conaty et al. (2018). As it turns out, the performance of this method is highly dependent of the initial distribution. As discussed before, the SPN with CATCLARA start only needs a couple of iterations to converge, while the SPN with Random start needs many more. One thing we noticed, especially for the booking data, is that CATCLARA creates a lot of small clusters only containing outliers. When

the SPN is learned, the part of the network describing these clusters is heavily overfitted. Therefore, these clusters do not change much between the iterations. The only instances that were assigned to a different cluster during the iterations were those from the larger (and thus less overfitted) clusters. The Random start generated random clusters of approximately equal sizes, which take many iterations to converge. But although most results (and thus most averages) of the SPN with Random start are worse than those of the SPN with CATCLARA start, interestingly enough for almost all testing datasets, the best of the 10 runs resulted in a higher accuracy than the SPN with CATCLARA start. Unfortunately, since in most real world applications (like our case study), there is no way to calculate the actual accuracy of the results, this will not help in finding the optimal solution. However, this does demonstrate that the current model is very dependent of the initial distribution. Because the used learning algorithm for the SPN is only one of many proposed methods, future research could be comparing various methods on their use for clustering.

Further, we presented our adapted version of CLARA that can handle categorical data, by modifying the categories so their values will be correctly interpreted by the distance function of the clustering algorithm. This is quite a simple adaptation, without even changing the distance function. However, more complex methods have been around. So, in combination with what we mentioned in the discussion of the testing experiments in section 5.4 about what variables should be considered categorical, this allows for future improvements by designing and implementing a more complex distance function for categorical variables.

Finally, considering the business perspective, here are some suggestions for future applications of the model. First of all, the data that was used for the case study can be enriched by adding variables that have to do with the steering of the flight and the strategy, since analysts are constantly influencing the behavior of the flight by steering it. This influence will then also be taken into account when determining if flights have similar behavior. Secondly, the model could be made forward looking in stead of only handling flights that have already flown. The SPN is learned using the backward looking data, but inference can be done of future flights. By being able to assign flights that are still in the booking process to any of the clusters, the analyst will get a better idea of the current booking status of the flight. On top of that, the model could be used to predict the missing part of the booking curve. A change that is needed for this, is the implementation of the alternative inference process that can handle missing data, using MAX nodes during inference. These applications will make the model more capable of supporting the analysts with their decision making.

A Testing results

A.1 Mean accuracy and standard deviation

Name	Type	*Random* Mean	*Random* Std	CLARA Mean	CLARA Std	CATCLARA Mean	CATCLARA Std
Labor	c	0.54561	0.04564	0.82456	-	0.80702	-
Postoperative	m	0.40889	0.02954	0.41111	-	0.46667	-
Zoo	m	0.25248	0.02251	0.72277	-	0.72277	-
MB promoters	c	0.53113	0.03178	0.56604	-	0.51887	-
Bridges	c	0.27664	0.01660	0.46729	-	0.47664	-
Lymph	c	0.31351	0.02592	0.41892	-	0.48649	-
Iris	n	0.38667	0.01937	0.90000	-	0.90000	-
Hepatitis	b	0.52968	0.02423	0.70323	-	0.70323	-
Wine	n	0.38034	0.01590	0.92135	-	0.92135	-
Autos	m	0.25756	0.00682	0.43902	-	0.41951	-
Glass	n	0.20514	0.00866	0.35047	-	0.35047	-
Audiology	c	0.16947	0.01001	0.30088	-	0.44690	-
Breast cancer	m	0.53077	0.02198	0.54196	-	0.52448	-
Hearth-h	c	0.23741	0.01387	0.39116	-	0.37415	-
Haberman	m	0.52484	0.01837	0.51961	-	0.51961	-
Dishonest users	m	0.52919	0.01509	0.53106	-	0.55590	-
Ecoli	n	0.18780	0.01349	0.59226	-	0.59226	-
Ionosphere	b	0.51538	0.01027	0.58405	-	0.58405	-
Colic	c	0.52663	0.01698	0.63587	-	0.73098	-
Vote	b	0.51908	0.01581	0.86437	-	0.86437	-
Cylinder	m	0.52463	0.01870	0.56481	-	0.58519	-
Credit approval	m	0.51103	0.01021	0.56049	-	0.81011	-
Diabetes	b	0.51432	0.00835	0.66797	-	0.66797	-
Solar flame	m	0.19737	0.00431	0.48311	-	0.43715	-
Car evaluation	c	0.26927	0.00619	0.27720	-	0.37963	-
Mushroom	c	0.50487	0.00401	0.70606	-	0.89537	-
Crowdsourced*	n	0.17605	0.00273	0.44116	-	0.44116	-
Letter*	b	0.05167	0.00047	0.20360	-	0.20360	-
Adult*	m	0.50198	0.00315	0.52218	-	0.60785	-

TABLE A.1: The mean and standard deviation of the accuracy over 10 runs for different methods for choosing the initial distribution. Datasets marked with (*) only had 5 runs because of their computational time. The results marked with red are significantly the best.

		0.1%		1%		2%	
Name	Type	Mean	Std	Mean	Std	Mean	Std
Labor	c	0.67895	0.06971	0.67544	0.07681	0.64211	0.06032
Postoperative	m	0.58556	0.05927	0.56333	0.07875	0.53667	0.08936
Zoo	m	0.48911	0.06735	0.56733	0.09611	0.52772	0.10248
MB promoters	c	0.63113	0.07010	0.58113	0.06402	0.58868	0.05322
Bridges	c	0.38131	0.04081	0.42150	0.05108	0.42991	0.07185
Lymph	c	0.56622	0.06882	0.53311	0.04643	0.55270	0.07509
Iris	n	0.61400	0.07387	0.62067	0.09745	0.62600	0.13468
Hepatitis	b	0.74387	0.08505	0.75032	0.08629	0.80065	0.02095
Wine	n	0.73933	0.12289	0.71910	0.12225	0.68315	0.10634
Autos	m	0.39220	0.07352	0.39317	0.07025	0.35659	0.07299
Glass	n	0.34626	0.03308	0.36495	0.02729	0.34019	0.03137
Audiology	c	0.32743	0.00939	0.32478	0.03422	0.31858	0.03741
Breast cancer	m	0.66538	0.08190	0.66469	0.07691	0.66364	0.05867
Hearth-h	c	0.50918	0.07483	0.53571	0.10846	0.57687	0.09398
Haberman	m	0.59020	0.03633	0.60065	0.06479	0.64412	0.06493
Dishonest users	m	0.68478	0.09411	0.60714	0.09165	0.62733	0.09719
Ecoli	n	0.53661	0.09358	0.49524	0.07545	0.52440	0.12949
Ionosphere	b	0.61595	0.04793	0.59459	0.07827	0.66410	0.06639
Colic	c	0.62255	0.02364	0.61902	0.01815	0.61250	0.02763
Vote	b	0.68736	0.15890	0.67839	0.11502	0.67816	0.13085
Cylinder	m	0.56167	0.03747	0.57704	0.00747	0.57167	0.03437
Credit approval	m	0.58637	0.09569	0.56493	0.06619	0.58897	0.06921
Diabetes	b	0.65247	0.02396	0.63268	0.02683	0.65117	0.02903
Solar flame	m	0.39953	0.06566	0.35216	0.03718	0.35328	0.05868
Car evaluation	c	0.38941	0.04077	0.38056	0.05338	0.41377	0.08448
Mushroom	c	0.69904	0.05589	0.65340	0.10380	0.68971	0.10916
Crowdsourced*	n	0.31721	0.05438	0.31313	0.05041	0.28465	0.01714
Letter*	b	0.12473	0.00318	0.13586	0.00855	0.11318	0.00730
Adult*	m	0.77037	0.01570	0.72319	0.01684	0.69851	0.01727

TABLE A.2: The mean and standard deviation of the accuracy over 10 runs of the SPN with a random start, for different values of α. Datasets marked with (*) only had 5 runs because of their computational time.

		0.1%		1%		2%	
Name	Type	Mean	Std	Mean	Std	Mean	Std
Labor	c	0.82456	-	0.82456	-	0.80702	-
Postoperative	m	0.58889	-	0.58889	-	0.37778	-
Zoo	m	0.73267	-	0.71287	-	0.71287	-
MB promoters	c	0.51887	-	0.50943	-	0.52830	-
Bridges	c	0.48598	-	0.49533	-	0.49533	-
Lymph	c	0.66216	-	0.50000	-	0.54054	-
Iris	n	0.90667	-	0.91333	-	0.91333	-
Hepatitis	b	0.72903	-	0.84516	-	0.78710	-
Wine	n	0.94944	-	0.95506	-	0.94382	-
Autos	m	0.48780	-	0.39512	-	0.40976	-
Glass	n	0.33178	-	0.34579	-	0.34112	-
Audiology	c	0.44690	-	0.44690	-	0.44690	-
Breast cancer	m	0.71329	-	0.66783	-	0.68881	-
Hearth-h	c	0.41837	-	0.50000	-	0.51361	-
Haberman	m	0.50980	-	0.58497	-	0.52941	-
Dishonest users	m	0.55590	-	0.55590	-	0.55590	-
Ecoli	n	0.61310	-	0.60714	-	0.57440	-
Ionosphere	b	0.60969	-	0.56125	-	0.58405	-
Colic	c	0.70109	-	0.69837	-	0.62772	-
Vote	b	0.87356	-	0.86667	-	0.85517	-
Cylinder	m	0.57593	-	0.57778	-	0.57593	-
Credit approval	m	0.81011	-	0.76263	-	0.57274	-
Diabetes	b	0.67708	-	0.68490	-	0.64974	-
Solar flame	m	0.38931	-	0.35647	-	0.35084	-
Car evaluation	c	0.41667	-	0.45891	-	0.46123	-
Mushroom	c	0.89599	-	0.89586	-	0.89623	-
Crowdsourced*	n	0.45624	-	0.44836	-	0.44049	-
Letter*	b	0.17105	-	0.15445	-	0.15485	-
Adult*	m	0.80154	-	0.77505	-	0.77107	-

TABLE A.3: The mean and standard deviation of the accuracy over 10 runs of the SPN with a CATCLARA start, for different values of α. Datasets marked with (*) only had 5 runs because of their computational time. The results marked with red are significantly the best.

B Cluster results from case study

B.1 Madrid

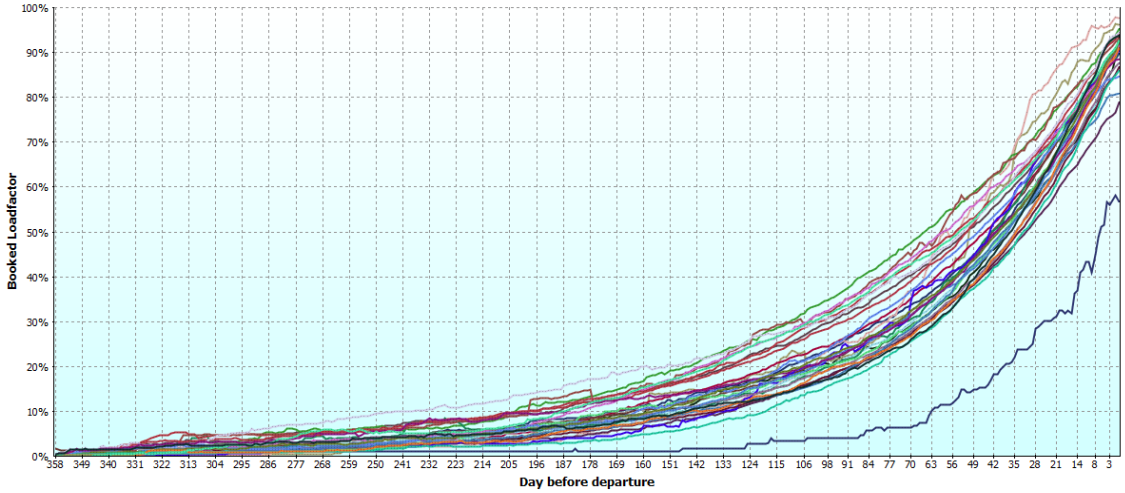

FIGURE B.1: The average booking curves of 30 clusters over all flights from dataset *Madrid2018*, clustered on flight profile.

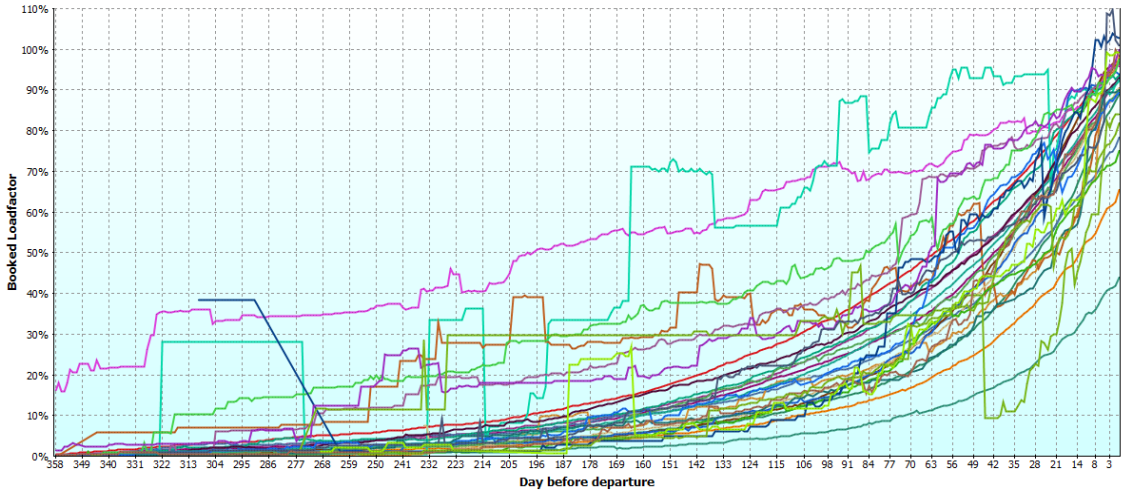

FIGURE B.2: The average booking curves of 30 clusters over all flights from dataset *Madrid2018*, clustered on booked load factor.

B.2 Barcelona

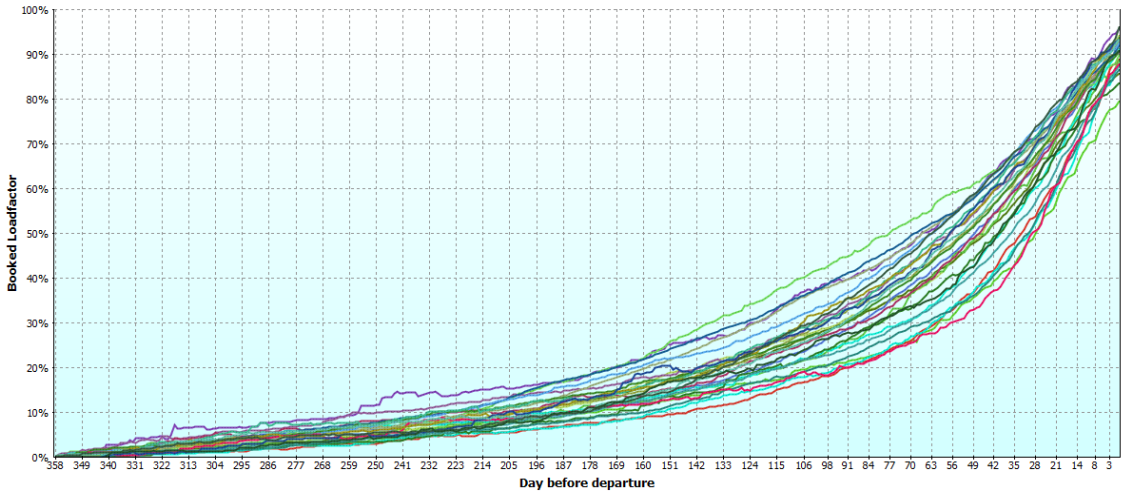

FIGURE B.3: The average booking curves of 30 clusters over all flights from dataset *Barcelona2018*, clustered on flight profile.

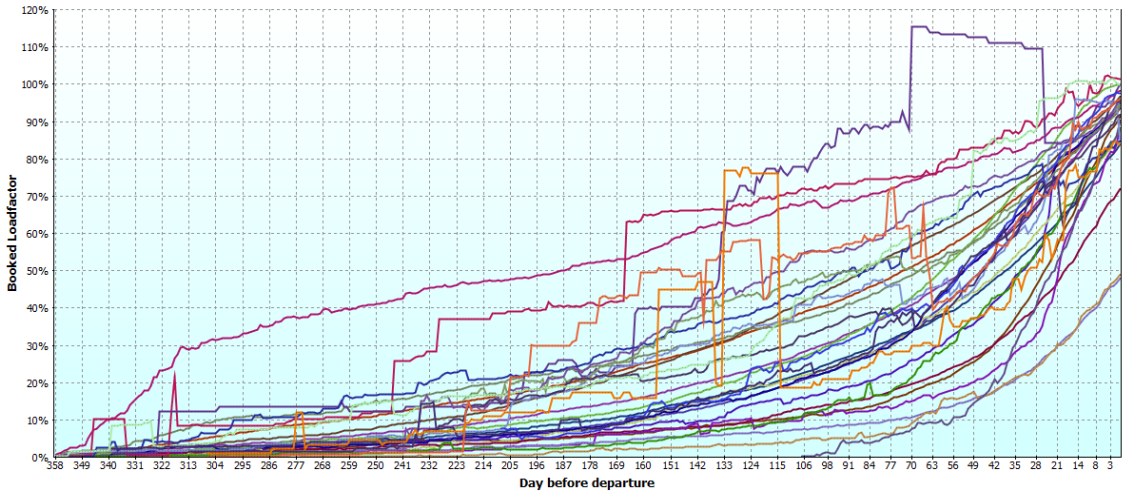

FIGURE B.4: The average booking curves of 30 clusters over all flights from dataset *Barcelona2018*, clustered on booked load factor.

B.3 SpainRest

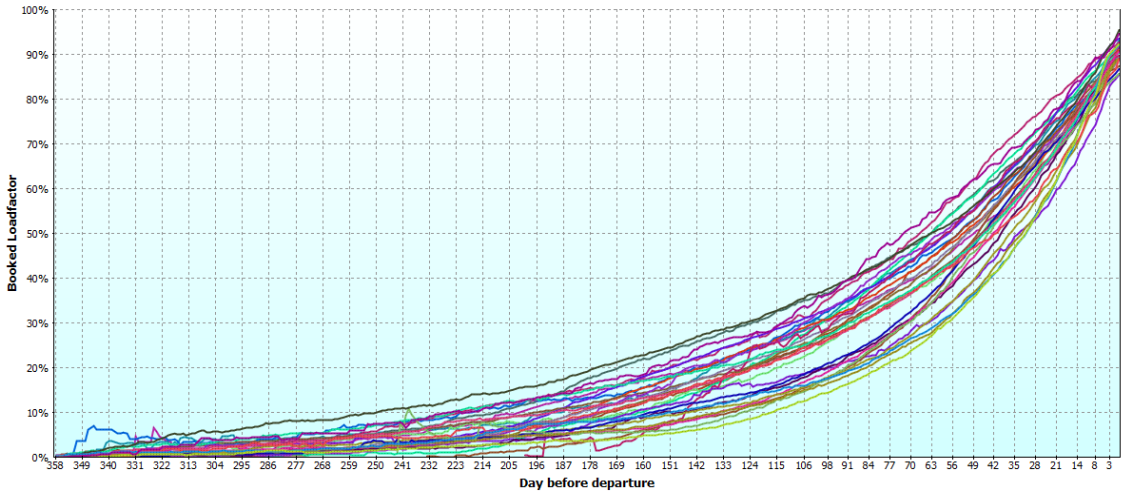

FIGURE B.5: The average booking curves of 30 clusters over all flights from dataset *SpainRest2018*, clustered on flight profile.

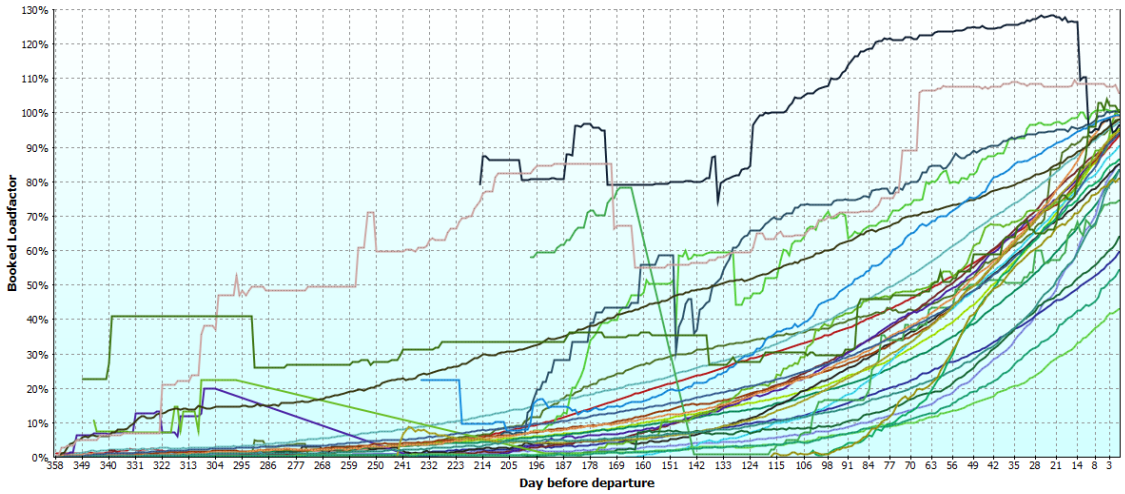

FIGURE B.6: The average booking curves of 30 clusters over all flights from dataset *SpainRest2018*, clustered on booked load factor.

B.4 Booking curves with distinct behavior

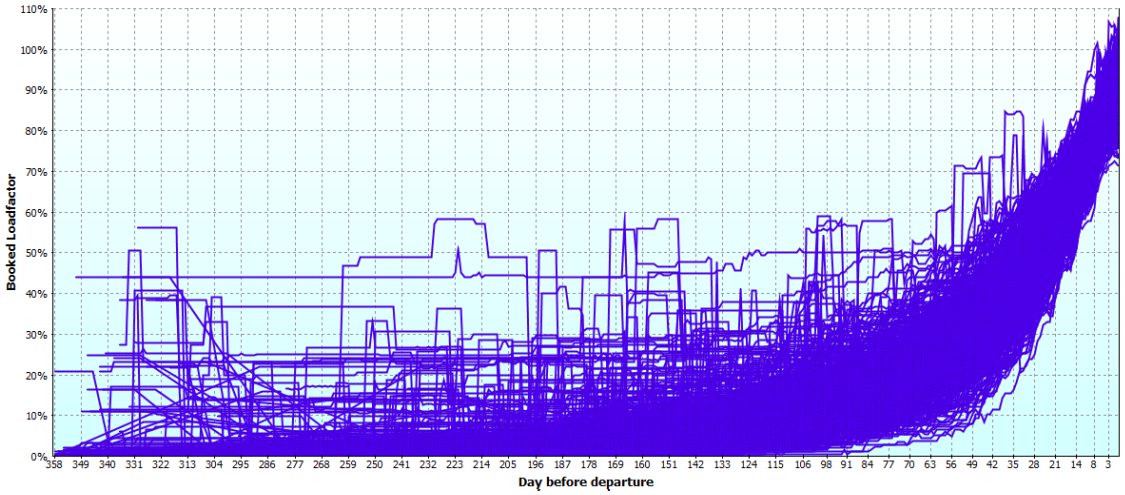

FIGURE B.7: All flights from a 250+ cluster from *Madrid2018* clustered on booked load factor, displaying a late booking behavior.

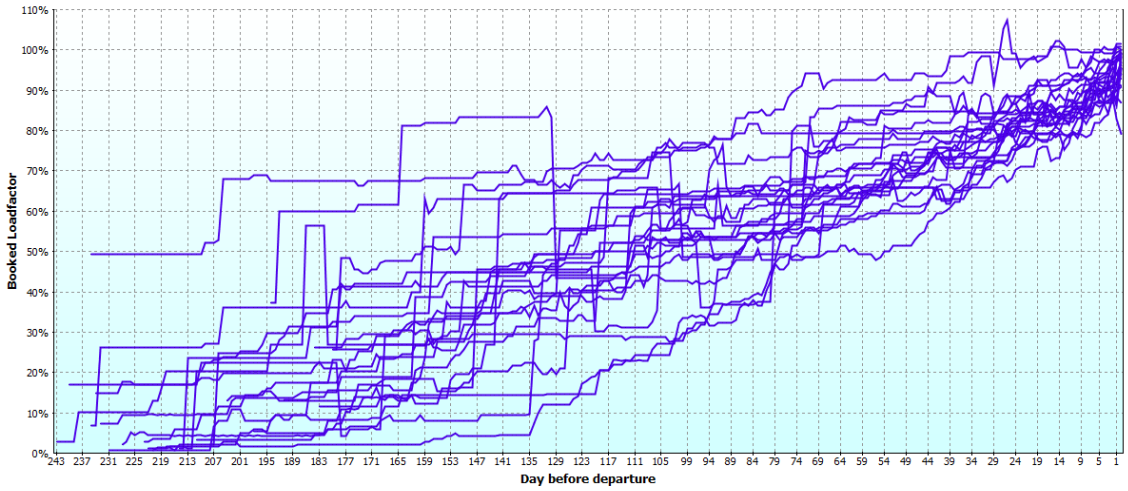

FIGURE B.8: All flights from a 25-49 cluster from *SpainRest2018* clustered on booked load factor, displaying early booking behavior.

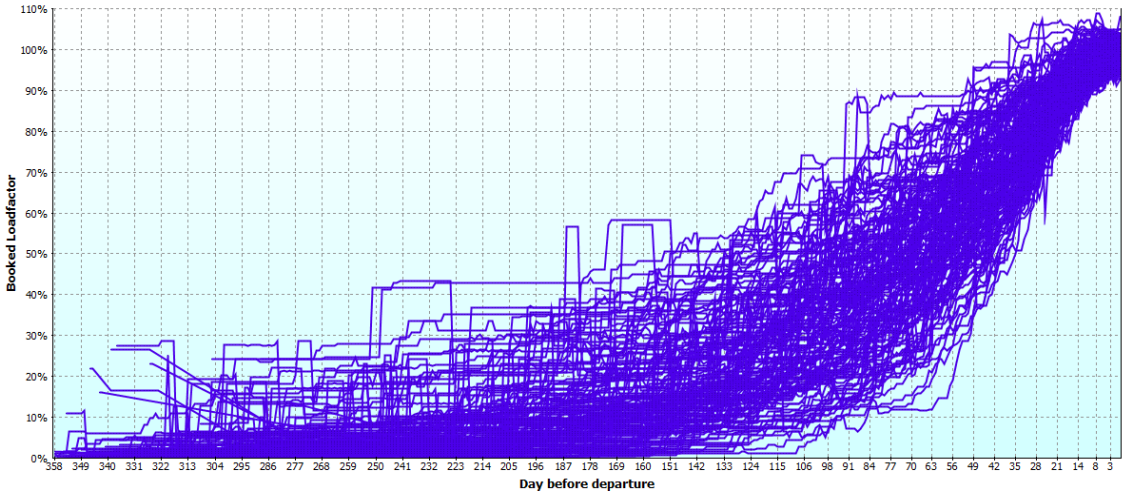

FIGURE B.9: All flights from a 250+ cluster from *Barcelona2018* clustered on booked load factor, displaying booking behavior that stagnates in the last days before departure.

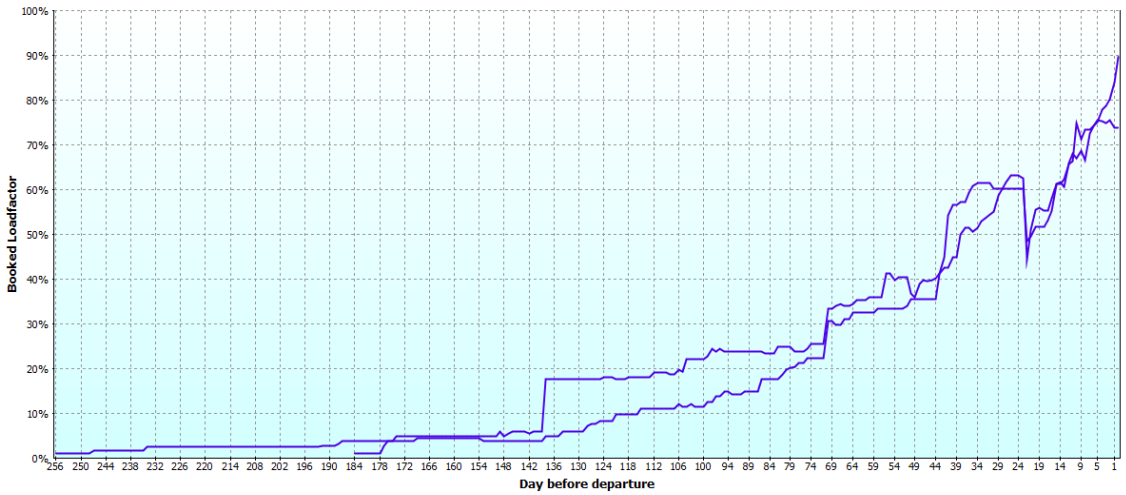

FIGURE B.10: Both flights from a 1-4 cluster from *Madrid2018* clustered on booked load factor, displaying a comparable booking behavior as the figure above, but including a major drop in booked load factor around 21 days before departure.

B.5 Special cases

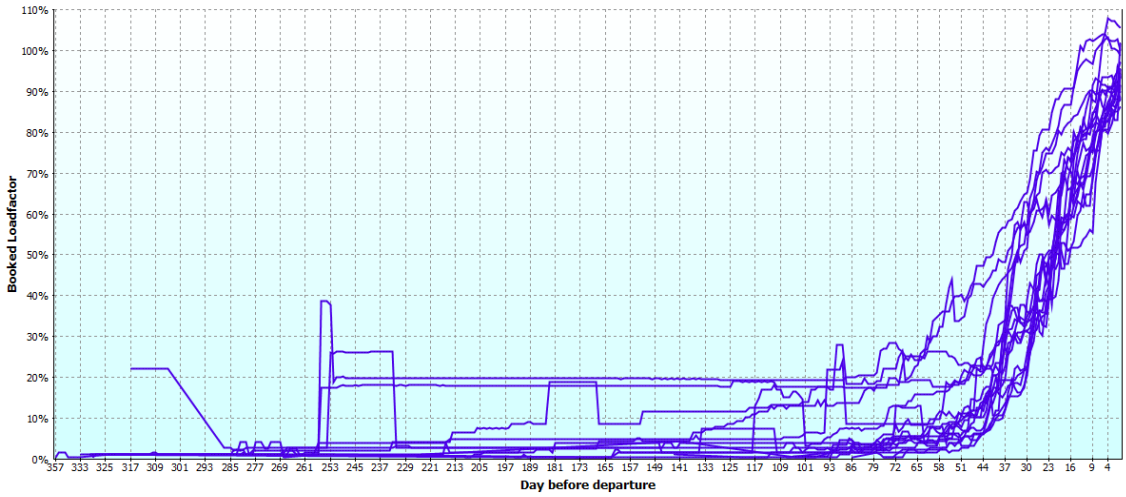

FIGURE B.11: Most flights from a 50-99 cluster from *Barcelona2018* are to or from Mobile World Congress in Barcelona, between 24-26 February and 1-2 March. The flights were clustered on cabin.

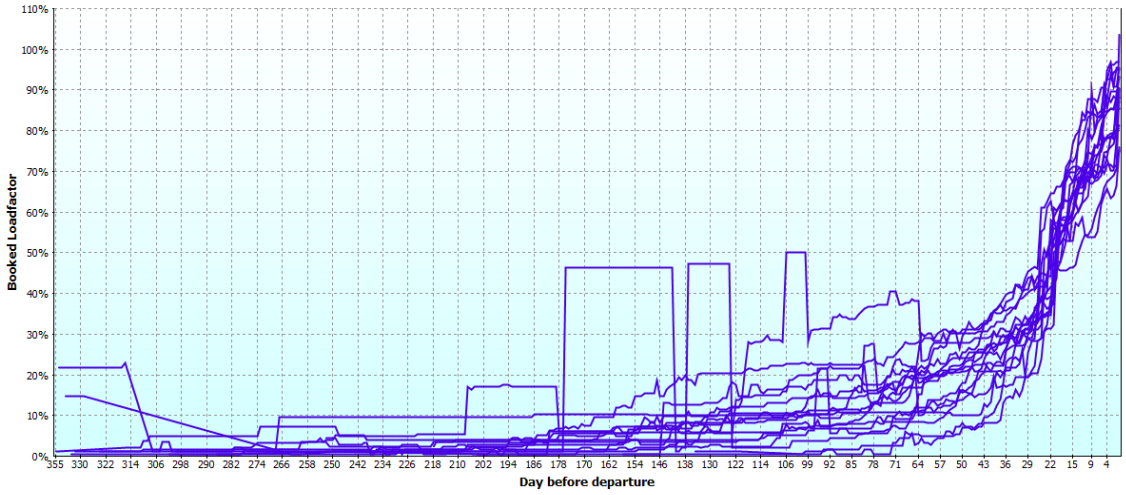

FIGURE B.12: All flights from a 25-49 cluster from *Barcelona2018* display a change in cabin capacity. Around 25 days before departure the aircraft type was changed, reducing the total capacity. Because the booked load factor is the number of bookings relative to the capacity, a strong increase is visible.

B.6 Traffic Type

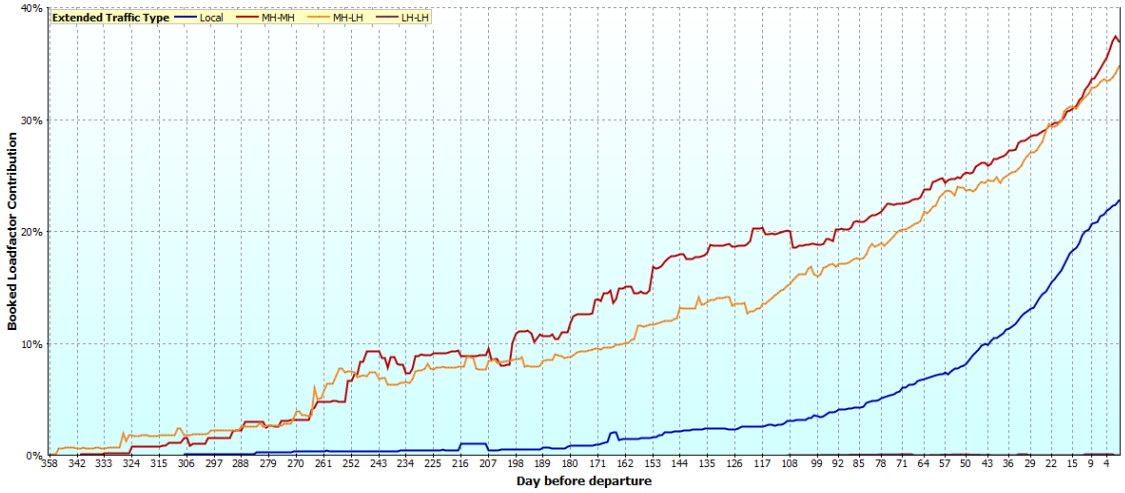

FIGURE B.13: The distribution of the traffic type for a cluster mostly containing flights that departed in bank 2.

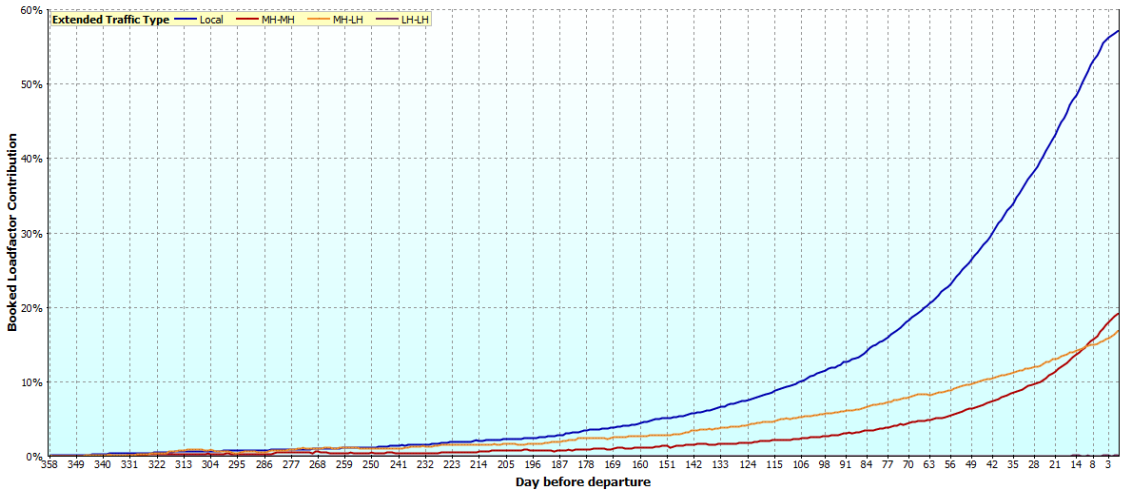

FIGURE B.14: The distribution of the traffic type for a cluster mostly containing flights that departed in bank 4 and 5.

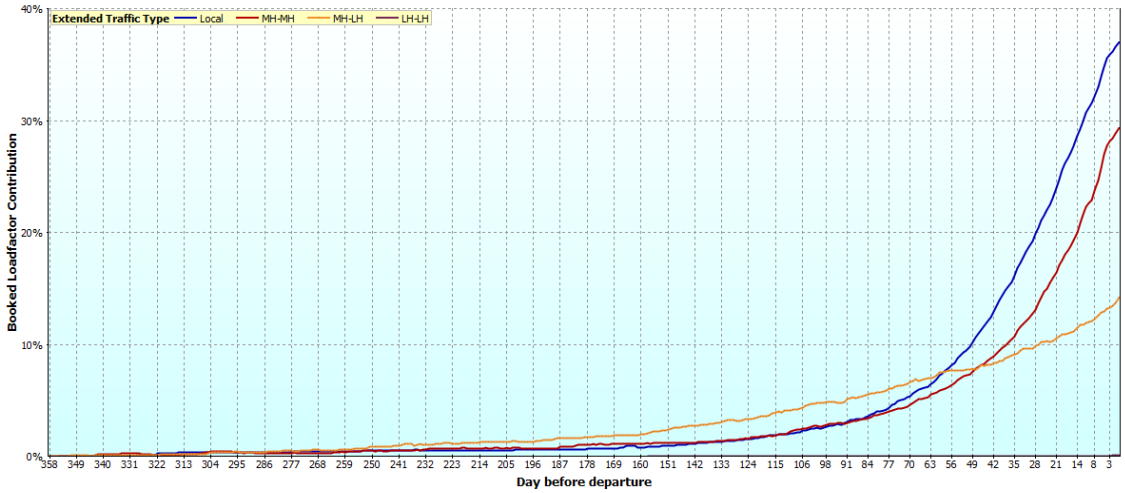

FIGURE B.15: The distribution of the traffic type for a cluster mostly containing flights that departed in bank 6.

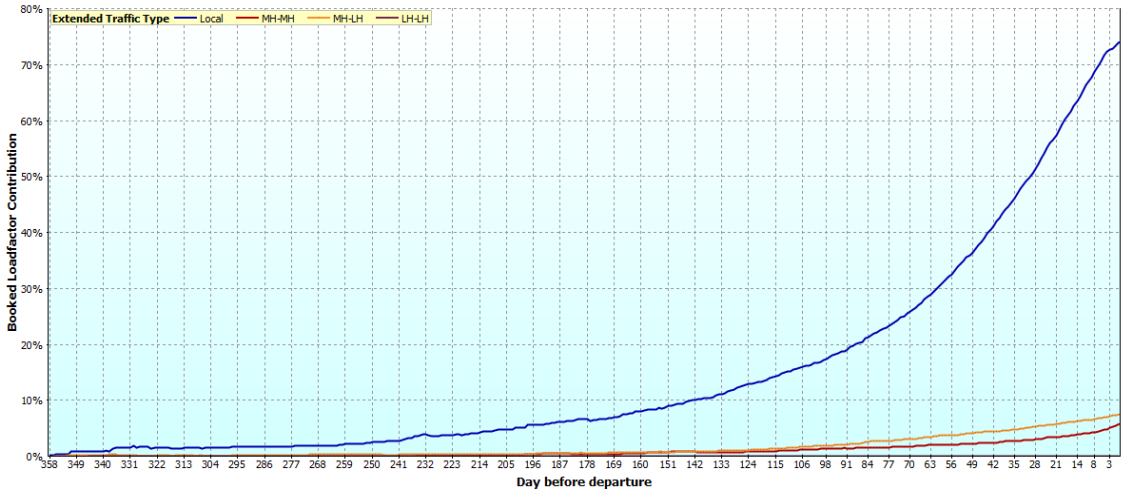

FIGURE B.16: The distribution of the traffic type for a cluster mostly containing flights that departed in bank 7.

C Code of the SPN in R

C.1 EM algorithm

```
spn.run <- function(data, name, maxi=10, nclasses, thr=0.01,
    random=TRUE, base=FALSE, catClara=TRUE, ncores=detectCores()/
    2) {

  data.acc <- c()
  for (i in 1:maxi) {
    accuracy <- spn.run.aux(data, nclasses=nclasses, thr=thr,
        random=random, base=base, catClara=catClara, ncores=ncores
        )

    data.acc <- c(data.acc, accuracy)
    }
  return(data.acc.perm)
}


spn.run.aux <- function(data, nclasses, classcol=ncol(data), thr
    =0.01, random=TRUE, catClara=TRUE, base=FALSE, ncores=
    detectCores()/2) {

  class <- data[,classcol]

  #Generate the initial class labels
  if (random) {
      data[,classcol] <- sample(1:nclasses, size=nrow(data),
          replace=TRUE)
  } else {
    if (catClara) {
      data[,classcol] <- CatClara(data[,-classcol], nclasses,
          samples=200, pamLike=TRUE, metric='manhattan')$
          clustering
    }
    else {
      data[,classcol] <- clara(data[,-classcol], nclasses,
          samples=200, pamLike=TRUE, metric='manhattan')$
          clustering
    }
  }

  #Recursively run SPN until convergence
  cnt <- 1
  if (!base) {
    maxcnt <- 20
    while (cnt < maxcnt) {
      spn = spn.learn(data, thr=thr, classcol=classcol)
      old <- data[,classcol]
```

```
      nr <- nrow(data)

      #do multithreading if a dimension of the dataset is larger
         than a certain size
      if(ncores > 1 && (nr > 1000 || nclasses > 3 || ncol(data) >
          30)) {
        expfunc <- c("spn.predict.single", "spn.predict", "spn.
           value", "spn.value.aux", "logsumexp")
        cl <- makeCluster(ncores)
        registerDoParallel(cl)
        clres <- foreach(i = 1:nr, .inorder=FALSE, .packages=c("
           cluster"), .combine=rbind, .export=expfunc) %dopar% {
          c(i,spn.predict.single(spn, data[i,,drop=FALSE],
             classcol=classcol))
        }
        stopCluster(cl)
        registerDoSEQ()
        clo <- order(clres[,1])
        for(i in 1:nr) {
          if(clres[clo[i],1] != i) stop("error collecting results
             ")
          data[i,classcol] <- clres[clo[i],2]
        }
      } else { #not multi-threaded
        for(i in 1:nr) {
          data[i,classcol] <- spn.predict.single(spn, data[i,,
             drop=FALSE], classcol=classcol)
        }
      }

      #test for convergence
      if (data.convergence(old, data[,classcol], nclasses=
         nclasses)) break

      cnt <- cnt + 1
    }
    if(cnt >= maxcnt) warning("Code has not converged within max
       iterations")
  }

  #calculate accuracy of clustering
  if (nclasses < 10)
    accuracy <- cluster.accuracy.perm(as.integer(class), data[,
       classcol], nclasses)
  else
    accuracy <- cluster.accuracy.hungarian(as.integer(class),
       data[,classcol], nclasses)

  return(accuracy)
}
```

C.2 CatClara algorithm

```
CatClara <- function(x, k, ncat=NULL, metric = "manhattan", stand
    = FALSE, samples = 5, sampsize = min(nrow(x), 40 + 2 * k),
   trace = 0, medoids.x = TRUE, keep.data = medoids.x, rngR =
   FALSE, pamLike = FALSE, correct.d = TRUE) {
```

```
  data <- x
  if(is.null(ncat)) {
    l <- spn.learncats(data)
    ncat <- l$ncat
    data <- l$data
  }

  #create total matrix and fill in -> sum over ncat
  if (metric != "jaccard")
    ncol <- sum(ncat) - sum(ncat == 2) + sum(ncat == 0)
  else
    ncol <- ncol(data)
  data.adapt <- data.frame(matrix(ncol=ncol,nrow=nrow(data)))

  cnt <- 1
  for (i in 1:length(ncat)) {
    if (ncat[i] > 2) {
      if (metric == "euclidean") {
        for (j in 1:ncat[i]) {
          data.adapt[cnt] <- (0.5 * (data[,i] == j) * sqrt(2))
          cnt <- cnt + 1
        }
      } else if (metric == "manhattan") {
        for (j in 1:ncat[i]) {
          data.adapt[cnt] <- ((data[,i] == j) * 0.5)
          cnt <- cnt + 1
        }
      } else if (metric == "jaccard") {
        ##Not implemented, just return normal data
        data.adapt[cnt] <- (data[,i] == j)
        cnt <- cnt + 1
      }
    } else {
      data.adapt[cnt] <- data[,i]
      cnt <- cnt + 1
    }
  }

  return(clara(data.adapt, k, metric, stand, samples, sampsize,
     trace, medoids.x, keep.data, rngR, pamLike, correct.d))
}
```

C.3 Learning algorithm for SPN

```
spn.learn <- function(data, ncat=NULL, thr=0.01, classcol=NULL,
   nclusters=2, root.sum=TRUE) {
  ## clean up of data and computation of number of categories per
      variable (unless already given)
  if(is.null(ncat)) {
    l <- spn.learncats(data,classcol=classcol)
    ncat <- l$ncat
    data <- l$data
  }
  ## in this implementation, the root node is a sum node related
     to the class variable (kind of a more discriminative
     approach), if that is given
```

```
  scope <- 1:ncol(data)
  root <- spn.learn.aux(data, ncat, scope=scope, thr=thr,
     nclusters=nclusters, classcol=classcol, last.prod=root.sum)
  return(new("spn",root=root,ncat=ncat))
}

spn.learncats <- function(data.ori, classcol=-1) {
  if(is.null(classcol)) classcol <- -1
  data <- matrix(0, nrow=nrow(data.ori), ncol=ncol(data.ori))
  ncat <- rep.int(0, ncol(data))
  for(i in 1:ncol(data)) {
    if (!is.factor(data.ori[1,i]) && i != classcol) {
      # considered Gaussian
      data[,i] <- data.ori[,i]
    } else {
      # considered nominal cats
      data[,i] <- as.integer(data.ori[,i])
      if (max(data[,i]) >= 1 || i == classcol) {
        ncat[i] <- max(data[,i])
      } else {
        stop('constant?\n')
      }
    }
  }
  return(list(data=data,ncat=ncat))
}
## learning is done recursively, splitting the data horizontally
   by clustering and vertically by "independence" tests
spn.learn.aux <- function(data, ncat, scope, thr, nclusters, last
   .prod=FALSE, classcol=NULL) {
    n <- length(scope)
    m <- nrow(data)

    if (n == 1) {
      ## single variable in the scope
      if(ncat[scope] > 1) {
        ## has it a single value? If so, then place an indicator
           function (as leaf node)
        if (length(unique(data[,scope]))==1) {
          return(new("node",scope=scope, value=data[1,scope],
             type=1,children=list(), len=0, size=1,id=round
             (10000000*runif(1)))) #'leaf-indicator'
        }
        ## if multiple values for the variable are still present
           in the data, then use a sum node with indicator
           functions as children
        ncategory <- ncat[scope]
          sumnode <- new("node",children=list(), scope=scope,
             weight=rep_len(0,ncategory), n=m, type=4, len=
             ncategory, size=ncategory+1,id=round(10000000*runif
             (1))) #'sum'

        for(i in 1:ncategory) {
          members <- which(data[,scope]==i)
          sumnode$children[[i]] <- new("node",scope=scope, value=
             i,  type=1,children=list(), len=0, size=1,id=round
             (10000000*runif(1))) #'leaf-indicator'
```

```
        sumnode$weight[i] <- length(members)+0.5
      }
      return(sumnode)
    } else {
      return(new("node",scope=scope, value=c(mean(data[,scope])
          ,max(1e-4,sd(data[,scope]),na.rm=TRUE)), type=2,
          children=list(), len=0, size=1,id=round(10000000*runif
          (1)))) #'leaf-gaussian'
    }
  } else if (n > 1) {
    if(!last.prod) { ## just to speed up, since never a
        product node will have a product node as child (easy
        to see that)
      ## let us build an undirected graph where two nodes (
          variables) are connected if they are dependent
      deplist <- list()
      depfunc <- function(i) { if(deplist[[i]] == i) return(i
          ) else return(deplist[[i]]); }
      for(i in 1:n) deplist[[i]] <- i
      for(i in 1:(n-1)) {
        for(j in (i+1):n) {
          fatheri <- depfunc(i)
          deplist[[i]] <- fatheri
          fatherj <- depfunc(j)
          deplist[[j]] <- fatherj
          if(fatheri != fatherj) {
            v <- 1
            unii <- length(unique(data[,scope[i]]))
            unij <- length(unique(data[,scope[j]]))
            if(unii > 1 && unij > 1) {
              if(m > 4 && ncat[scope[i]] == 0 && ncat[scope[j
                  ]] == 0) {
              ## both continuous
                v <- suppressWarnings(cor.test(data[,scope[i
                    ]],data[,scope[j]],method='kendall'))$p.
                    value
              }
              if(m > 4*unij && ncat[scope[i]] == 0 && ncat[
                  scope[j]] > 1) {
              ## i continuous, j discrete
                v <- suppressWarnings(kruskal.test(data[,
                    scope[i]],data[,scope[j]]))$p.value
              }
              if(m > 4*unii && ncat[scope[i]] > 1 && ncat[
                  scope[j]] == 0) {
              ## i discrete, j continuous
                v <- suppressWarnings(kruskal.test(data[,
                    scope[j]],data[,scope[i]]))$p.value
              }
              if(m > unii*unij*2 && ncat[scope[i]] > 1 &&
                  ncat[scope[j]] > 1) {
              ## both discrete
                v <- suppressWarnings(chisq.test(data[,scope[
                    i]],data[,scope[j]]))$p.value
              }
              if(v < thr)
                deplist[[fatherj]] <- fatheri
```

```
            }
          }
        }
      }
      clu <- list(no=0,membership=1:n)
      for(i in 1:n)
        clu$membership[i] <- depfunc(i)
      clu$unique <- unique(clu$membership)
      clu$no <- length(clu$unique)
      for(i in 1:n)
        clu$membership[i] <- which(clu$membership[i] == clu$
            unique)

      ## and from such graph, take the components to form the
          children of the product node, as long as there are
         more than one component (single component means that
          we cannot split vertically at this moment, and in
         that case we move on to split horizontally
      if(clu$no > 1) {
        prodnode <- new("node",children=list(), scope=scope,
            n=m, type=3, len=clu$no, size=1,id=round(10000000*
            runif(1))) #'prod'
        sizes <- rep_len(0,clu$no)
        children <- list()
        for(i in 1:clu$no) {
          children[[i]] <- spn.learn.aux(data, ncat, scope=
              scope[which(clu$membership == i)], thr=thr,
              nclusters=nclusters, last.prod=TRUE, classcol=
              classcol)
          sizes[i] <- children[[i]]$size
        }
        prodnode$size <- 1 + sum(sizes)
        o <- order(sizes)
        for(i in 1:clu$no)
          prodnode$children[[i]] <- children[[o[i]]]

        return(prodnode)
      }
    }
}

## we were not able to cut vertically, so we run clustering
    of the data (each row is a point), and then we use the
    result of clustering to create the children of a sum node
sumnode <- new("node",children=list(), scope=scope, weight=
    vector(), n=m, len=0, type=4, size=1,id=round(10000000*
    runif(1))) #'sum'
nrchildren <- nclusters

# Creating sum node with class as cluster
if(!is.null(classcol) && !(classcol %in% scope))
    classcol <- NULL
if(!is.null(classcol) && ncat[classcol] > 1) {
    clu.ind <- data[,classcol]
    nrchildren <- ncat[classcol]
    classcol <- NULL
} else {
```

```
      if(nclusters >= nrow(data))
        clu.ind <- 1:nrow(data)
      else {
        clu.ind <- CatClara(data[,scope,drop=FALSE],nclusters,
            ncat=ncat[scope],samples=20,pamLike=TRUE,metric='
            manhattan')$clustering
      }
    }

    j <- 1
    ## after clusters are found, build the children using that
       partition of the data
    for(i in 1:nrchildren) {
      members <- which(clu.ind == i)
      if(length(members) > 0) {
        sumnode$children[[j]] <- spn.learn.aux(data[members,,drop
            =FALSE], ncat, scope=scope, thr=thr, nclusters=
            nclusters, last.prod=FALSE, classcol=classcol)
        sumnode$weight <- c(sumnode$weight, length(members))

        j <- j + 1
      }
    }

    for(ch in 1:length(sumnode$children)) {
      sumnode$size <- sumnode$size + sumnode$children[[ch]]$size
    }
    sumnode$len <- length(sumnode$children)
    return(sumnode)
}
```

C.4 Inference algorithm for SPN

```
spn.predict.single <- function(spn, data, classcol=ncol(data)) {
  max.class <- spn.predict(spn, data, classcol=classcol)[2]
  return(max.class)
}

spn.predict <- function(spn, data, classcol=ncol(data)) {
  nclass <- spn$ncat[classcol]
  res <- c()
  for(i in 1:nrow(data)) {
    nclass <- spn$ncat[classcol]
    res <- c()
    cfg <- list()
    cfg$scope <- 1:ncol(data[i,])
    cfg$value <- as.vector(data[i,],"numeric")
    maxclass <- 1
    maxlogpr <- -Inf

    for(j in 1:nclass) {
      cfg$value[classcol] <- j
      logpr <- spn.value(spn, cfg)
      if(logpr > maxlogpr) {
        maxclass <- j
        maxlogpr <- logpr
      }
```

```
    }
    res <- rbind(c(data[i,classcol], maxclass, exp(maxlogpr)))
  }
  return(res)
}

spn.value <- function(spn, config) {
  ## get the answer recursively
  return(spn.value.aux(spn$root, config))
}

spn.value.aux <- function(node, config) {
    if(node$type == 1) { #'leaf-indicator'
      ## for leaf nodes, return log(1) unless the var of this
         leaf appears in the config and is not compatiable with
         it
      pos <- which(node$scope == config$scope)
      v <- 0
      if(length(pos) > 0 && config$value[pos] != node$value)
        v <- -Inf ## log(0)
      return(v) ## log(1)
  }
  else if(node$type == 2) { #'leaf-gaussian'
    ## for leaf nodes, return log(1) unless the var of this leaf
       appears in the config
    pos <- which(node$scope == config$scope)
    if(length(pos) > 0) {
      v <- dnorm(config$value[pos],mean=node$value[1],sd=node$
         value[2],log=TRUE)
      return(v)
    }
    return(0) ## log(1)
  }
  else if(node$type == 3) { #'prod'
    ## for product nodes, return the sum of the result of the
       children (sum since they are logs)
    lc <- node$len
    val <- 0
    for(nod in 1:lc) {
        val <- val + spn.value.aux(node$children[[nod]], config)
           ## log product is sum of logs
        if (val == -Inf) break
    }
    return(val)
  }
  else if(node$type == 4) { #'sum'
    ## for sum nodes, combine the results from the children with
       the appropriate weights
    lc <- node$len
    vals <- rep_len(0,lc)
    w <- node$weight/sum(node$weight)
    for(i in 1:lc) {
        res <- spn.value.aux(node$children[[i]], config)
        vals[i] <- log(w[i]) + res
    }
    ## weird case: only one child, then return its value
    if(lc == 1) return(vals)
```

```
    ## since vals are logs, we need to combine them with log-sum-
       exp
    v <- logsumexp(vals)

    return(v)
  }
}

logsumexp <- function(x) {
  p <- which.max(x)
  if(x[p] == -Inf) return(-Inf)
  return(log1p(sum(exp(x[-p] - x[p]))) + x[p])
}
```

C.5 Convergence algorithm for SPN

```
data.convergence <- function(old, new, nclasses, thr = 0.01) {
  if (nclasses < 10)
    return(cluster.accuracy.perm(old, new, nclasses) > 1.0 - thr)
  else
    return(cluster.accuracy.hungarian(old, new, nclasses) > 1.0 -
        thr)
}

cluster.accuracy.perm <- function(real, pred, nclass) {
  # generate list of all permutations of nclass
  permutations <- t(array(unlist(permn(1:nclass)), dim=c(nclass,
     gamma(nclass+1))))

  max <- -Inf
  for(p in 1:nrow(permutations)) {
    pred.perm <- c()
    for (i in 1:nclass)
      pred.perm[pred == i] <- permutations[p, i]
    sum <- sum(real == pred.perm)
    if (sum > max) max <- sum
  }
  return((max/length(real)))
}

cluster.accuracy.hungarian <- function(real, pred, nclass) {
    # generate contingency matrix of fixed size nclass x nclass
    contingency <- matrix(0, nrow=nclass, ncol=nclass)

    for (i in 1:length(real))
        contingency[real[i], pred[i]] <- contingency[real[i],
           pred[i]] + 1
    perm <- solve_LSAP(as.table(contingency), max=TRUE)
    pred.perm <- c()
    for (i in 1:nclass)
        pred.perm[pred == perm[i]] <- i
    sum <- sum(real == pred.perm)

    return((sum/length(real)))
}
```

Bibliography

A. Abdullin and O. Nasraoui. Clustering heterogeneous data sets. In *Web Congress (LA-WEB), 2012 Eighth Latin American*, pages 1–8. IEEE, 2012.

A. Ahmad and L. Dey. A *k*-mean clustering algorithm for mixed numeric and categorical data. *Data & Knowledge Engineering*, 63(2):503–527, 2007.

P. Berkhin. Survey of clustering data mining techniques, 2002. *Accrue Software: San Jose, CA*, 2004.

W. C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai. Language modeling with sum-product networks. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

D. Conaty, J. M. Del Rincon, and C. P. De Campos. Cascading sum-product networks using robustness. In *International Conference on Probabilistic Graphical Models*, pages 73–84, 2018.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

A. Dennis and D. Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, pages 2033–2041, 2012.

V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS-clustering categorical data using summaries. In *KDD*, volume 99, pages 73–83, 1999.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880, 2013.

A. Hinterhuber and S. M. Liozu. The micro-foundations of pricing, 2017.

Z. Huang. Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD)*, pages 21–34. Singapore, 1997.

M. Irwin. Short-term revenue forecasting at klm. Master's thesis, Leiden University, 2010.

A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8): 651–666, 2010.

L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. A Wiley-Interscience publication. Wiley, 1990.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

M. Lázaro, I. Santamarıa, and C. Pantaleón. A new EM-based training algorithm for RBF networks. *Neural Networks*, 16(1):69–77, 2003.

C. K. Lee, T. D. Lin, and C. H. Lin. Pattern analysis on the booking curve of an intercity railway. *Journal of the Eastern Asia Society for Transportation Studies*, 6:303–317, 2005.

D. Li, J. Deogun, W. Spaulding, and B. Shuart. Towards missing data imputation: a study of fuzzy k-means clustering method. In *International conference on rough sets and current trends in computing*, pages 573–579. Springer, 2004.

W. Lohmann. Strategy in seat inventory control: an empirical research at klm on improving initial steering strategies. Master's thesis, University of Twente, 2011.

P. Luo, X. Wang, and X. Tang. A deep sum-product architecture for robust facial attributes analysis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2864–2871, 2013.

M. Ma, J. Liu, and J. Cao. Short-term forecasting of railway passenger flow based on clustering of booking curves. *Mathematical Problems in Engineering*, 2014, 2014.

D. D. Mauá, F. G. Cozman, D. Conaty, and C. P. De Campos. Credal sum-product networks. In *Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications*, pages 205–216, 2017.

D. D. Mauá, D. Conaty, F. G. Cozman, K. Poppenhaeger, and C. P. De Campos. Robustifying sum-product networks. *International Journal of Approximate Reasoning*, 101: 163–180, 2018.

J. I. McGill and G. J. Van Ryzin. Revenue management: Research overview and prospects. *Transportation science*, 33(2):233–256, 1999.

T. K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

F. V. Nelwamondo, S. Mohamed, and T. Marwala. Missing data: A comparison of neural network and expectation maximization techniques. *Current Science*, pages 1514–1521, 2007.

R. Peharz. *Foundations of sum-product networks for probabilistic modeling*. PhD thesis, Aalborg University, 2015.

R. Peharz, S. Tschiatschek, F. Pernkopf, and P. Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752, 2015.

H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

M. Ratajczak, S. Tschiatschek, and F. Pernkopf. Sum-product networks for structured prediction: Context-specific deep conditional random fields. In *Proc Workshop on Learning Tractable Probabilistic Models*, volume 1, pages 1–10, 2014.

S. Salvador and P. Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 576–584. IEEE, 2004.

E. Segal, A. Battle, and D. Koller. Decomposing gene expression into cellular processes. In *Biocomputing 2003*, pages 89–100. World Scientific, 2002.

B. M. Sguerra and F. G. Cozman. Image classification using sum-product networks for autonomous flight of micro aerial vehicles. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 139–144. IEEE, 2016.

T. Svrcek. Modeling airline group passenger demand for seat inventory control. *Presentations from the 1992 MIT/industry cooperative research program annual meeting*, pages 58–75, 1992.

R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

L. R. Weatherford. Intelligent aggressiveness: Combining forecast multipliers with various unconstraining methods to increase revenue in a global network with four airlines. *Journal of Revenue and Pricing Management*, 14(2):84–96, 2015.

L. R. Weatherford and P. P. Belobaba. Revenue impacts of fare input and demand forecast accuracy in airline yield management. *Journal of the Operational Research Society*, 53(8):811–821, 2002.

L. R. Weatherford and S. Pölt. Better unconstraining of airline demand data in revenue management systems for improved forecast accuracy and greater revenues. *Journal of Revenue and Pricing Management*, 1(3):234–254, 2002.

H. Zhang, P. Xie, and E. Xing. Missing value imputation based on deep generative models. *arXiv preprint arXiv:1808.01684*, 2018.