Institute of Statistics, Biostatistics and Actuarial Sciences

Clustering with variational autoencoders

Thesis committee :
Prof. Donatien Hainaut *Supervisor*
Prof. Johan Segers *Reader*
Prof. Christian Hafner *President*

Thesis submitted in partial
fulfillment for the degree of
Master in Data Science : Statistic

by : Chloé Marchal

Louvain-La-Neuve
June 2024

Acknowledgements

I would like to thank my family and friends for their support and belief in me throughout this thesis and the entire course of my studies. Additionally, I wish to thank my supervisor, Donatien Hainaut, for his help, guidance, and patience.

Contents

List of Figures

List of Tables

1 Introduction

In the machine learning field, deep generative models have been widely used in the past years in order to cope with the problem of data scarcity in certain fields (Jordan & Mitchell, 2015). The two most known techniques for data generation are the Variational Autoencoders (Kingma & Welling, 2013) and the Generative Adversarial Networks (Goodfellow et al., 2014). Both approaches use generative models. GANs have a tendency to create images that are of better quality to the human eyes while VAEs model the density of the data more accurately according to the likelihood criterion but with some blurriness of the outputs (Singh & Ogunfunmi, 2021). Our attention will be drawn towards the VAEs and their bottleneck architecture, which gives the opportunity to discover potential underlying structure in high-dimensional data.

To provide a definition of the VAEs, they are neural networks, more precisely a Bayesian extension of the conventional autoencoders. They are used for dimensionality reduction and for embedding categorical variables since they are composed of an encoder and a decoder with a bottleneck in between.

The encoder allows to map data into a lower-dimensional space, and the decoder transforms the embeddings back to the original space while minimizing the reconstruction error. For this purpose, a distribution over the input data is learned and new data is sampled from that distribution. With such an approach, latent variables are hoped to be less correlated, disentangled, giving a better representation of the hidden structure of the data.

Although results using the vanilla VAE are satisfactory, variants have been introduced in order to improve the disentanglement of the representation. Additional challenges remain as much on a theoretical than on a practical level (Asperti et al., 2021; Singh & Ogunfunmi, 2021). These include issues like the blurriness of the generated outputs, some variance loss and a balancing issue between good latent representations or high quality outputs.

On the application side, VAEs have been widely implemented using images, speech/ audio and text data, however research and application on other types of data is lacking. Singh & Ogunfunmi (2021) have explored the possibilities VAEs can offer when applied on times series and signal processing in domains such as finance, speech source seperation and bio-signals. Also Jamotton & Hainaut (2023) have discussed the performance of VAEs to generate fake insurance policies. This is relevant in the insurance sector with the introduction of the recent GDPR regulations, but other sectors facing similar regulatory challenges might be interested in this approach as well. Furthermore, an equivalent process

was utilized by Salim (2018) on medical data creating new patients which would offer the possibility to increase the quantity of data that is sometimes missing in the medical world.

Knowing this, we can summarize the current research around this method into three poles of research (Wei et al., 2020) :

- find solutions to reduce the entanglements of the VAEs;
- applications and customization of the VAEs to real-world data;
- reduce the blurriness of generated images.

In this thesis, our focus will be on the second aspect : we will implement a variational autoencoder and conduct analyses on real-world data. Our goal is to explore the possibilities VAEs offer in terms of representations and data generation. Specifically, we will use the dimensionality reduction feature of the model in order to encode categorical variables to then find some hidden structure in the data. Those embeddings will further serve for clustering using K-means which will allow us to see if the variational autoencoder helps to detect meaningful clusters. The obtained clusters could represent the main profiles of our dataset. Those results will be compared with a second encoding technique we will use to perform clustering : the Burt's matrix.

In a second phase, we will utilize the latent variables to perform prediction using a regression model—the generalized linear model. This way, we will compare the predictive power of the clustering and the regression. Finally, we'll generate new customer profiles and conduct analyses on them.

The chosen dataset (Airline Passenger Satisfaction) is a compilation of reviews of passengers of a US airline. The airline industry is highly competitive and understanding customers in order to deliver a higher quality of service could increase their satisfaction. A higher satisfaction may result in more trust, loyalty from customers but also leading them to recommend an airline more easily (Park et al., 2022; An & Noh, 2009).
Research on airline passengers' satisfaction using deep learning is limited. Most studies conducted in this field are using the classical statistical techniques such as regressions, factorial analyses and so on which all assume linear relationships. Park et al. (2022) proposed an overview and comparison of various deep learning techniques aiming at predicting customer churn and satisfaction. Furthermore, a study conducted by Aldunate et al. (2022) tried to understand the factors that boost customer satisfaction by using responses to open survey questions. Deep learning techniques were thus applied in the context of the study to process textual data.

With this thesis, we will explore the possibilities of performing deep clustering on such data and see how it could help to understand and distinguish main profiles of passengers and their predicted satisfaction. This information could further be used to detect points of improvement in the service for customers that are unsatisfied and maybe develop strategies to increase the satisfaction.

This work is organized as follows. Section 2 presents the dataset with some descriptive analyses and starts the preprocessing of the data through missing values treatment, outliers detection and encoding. This is followed by the theoretical explanations about the model of main interest : the variational autoencoders in section 3, and some additional learning techniques in section 4, namely the K-means algorithm, the t-distributed Stochastic Neighbor Embedding and the generalized linear model. Section 5 covers the case study with the actual implementation of the models with the choice of the hyperparameters and the architecture of the models. The results are also presented in that section and further discussed in the last section, section 6, which concludes the thesis.

2 Presentation of the dataset

In this first section, we will have a look at the dataset that was chosen for the case study of this thesis (Airline Passenger Satisfaction). As said earlier, the dataset gathers information about the satisfaction of airline clients alongside some personal information about passengers and additional details such as the level of service in the flight, potential departure and arrival delay, and so on. This dataset is free of access on Kaggle and is composed of 23 variables from which 4 are continuous and 19 are categorical. There were two additional variables in the initial dataset (*id*, *Unnamed: 0*), but they are of no use; thus, we drop them directly and won't discuss them further.

Here is an exhaustive list of the variables present in the dataset :

- Gender (categorical): corresponds to the gender of the passenger (Male or Female);
- Customer Type (categorical): indicates how loyal the customer is (Loyal customer or Disloyal customer);
- Type of Travel (categorical): denotes the purpose of the travel (Personal Travel or Business Travel);
- Class (categorical): indicates the class in which the passenger decided to travel (Business, Eco or Eco Plus);
- Inflight wifi service (categorical): satisfaction level of Inflight wifi service (0: Not offered in flight ; 1-5: ratings out of 5);
- Departure/Arrival time convenient (categorical): satisfaction level of Departure/ Arrival time convenient (0: Not offered in flight ; 1-5: ratings out of 5);
- Ease of Online booking (categorical): satisfaction level of Online booking (0: Not offered in flight ; 1-5: ratings out of 5);
- Gate location (categorical): satisfaction level of Gate location (0: Not offered in flight ; 1-5: ratings out of 5);
- Food and drink (categorical): satisfaction level of Food and drink (0: Not offered in flight ; 1-5: ratings out of 5);
- Online boarding (categorical): satisfaction level of Online boarding (0: Not offered in flight ; 1-5: ratings out of 5);
- Seat comfort (categorical): satisfaction level of Seat comfort (0: Not offered in flight ; 1-5: ratings out of 5);
- Inflight entertainment (categorical): satisfaction level of Inflight entertainment (0: Not offered in flight ; 1-5: ratings out of 5);

- On-board service (categorical): satisfaction level of On-board service (0: Not offered in flight ; 1-5: ratings out of 5);
- Leg room service (categorical): satisfaction level of Leg room service (0: Not offered in flight ; 1-5: ratings out of 5);
- Baggage handling (categorical): satisfaction level of Baggage handling (0: Not offered in flight ; 1-5: ratings out of 5);
- Check-in service (categorical): satisfaction level of Check-in service (0: Not offered in flight ; 1-5: ratings out of 5);
- Inflight service (categorical): satisfaction level of Inflight service (0: Not offered in flight ; 1-5: ratings out of 5);
- Cleanliness (categorical): satisfaction level of Cleanliness (0: Not offered in flight ; 1-5: ratings out of 5);
- Satisfaction (categorical): airline satisfaction level (Satisfaction, Neutral or dissatisfaction);
- Age (continuous): the passenger's age;
- Flight distance (continuous): the distance the plane traveled during the trip (in km);
- Departure Delay in Minutes (continuous): Minutes delayed from the departure time;
- Arrival Delay in Minutes (continuous): Minutes delayed from the arrival time.

In this dataset, we lack information about the company operating the flight, departure and arrival locations, and the ticket price. These pieces of information could be valuable for predicting passenger satisfaction. For instance, it would be interesting to investigate if passengers flying low-cost companies are more likely to be satisfied with their experience on the flight. The level of service in such companies is often lower but the passengers book low-cost flights with prior knowledge of the low level of service. We could thus wonder if the satisfaction for those low-cost carriers is higher than for conventional companies offering a similar quality of service. For example, Riorini & Widayati (2018) conducted a research about satisfaction of customers in low-cost companies and found that satisfaction can be correlated to the ticket price. A fairer price can lead to more satisfied customers.

The departure and arrival locations are also important. Maybe the quality of service at the airport itself impacts passengers' perception, which directly affects the satisfaction rating of the passengers. It could thus provide valuable information, and with this, some airports might appear to be correlated to unsatisfied customers, or inversely, some airports might be offering excellent assistance leading to more satisfied customers.

2.1 Exploratory analysis

2.1.1 Distributions

By taking a quick look at the distribution of the different variables, we can make hypotheses about the dominant profiles that would emerge when we perform clustering. In this section, we will also highlight general insights about the dataset. The graphs supporting our analysis can be found in the appendix, section 6.1. We use simple barplots in combination with plots where each category of the *satisfaction* variable is represented by a different color. Note that variables *Arrival delay in Minutes* and *Departure delay in Minutes* have already been categorized for better representation in the barplots. They are thus presented by the variables *ArrivalDelayCat* and *DepartDelayCat* respectively. The variables *Age* and *FlightDistance* will also be categorized, but they are displayed as continuous on the histograms. The categorization is explained in section 2.2.2.

Now, it is important in classification problems that the target variable is balanced. Indeed, if we had a category with 90% of the observations and the other with the remaining 10%, it would be safe for the algorithm to always predict a high probability for an individual to belong to the first class. With such predictions, the algorithm would still manage to achieve good performance. We see in our case that *satisfaction* is fairly divided with approximately 70,000 individuals who are neutral or dissatisfied and more than 50,000 who are satisfied. When mapped into 0 (neutral or dissatisfied) and 1 (satisfied), we get an average satisfaction of 0.4344, which leans towards the less satisfied side.

Moreover, the target is binary : the passenger is either satisfied or neutral/dissatisfied. This needs to be taken into account in the implementation part for choices such as the loss function.

A balanced distribution is also important for the other variables. If any categorical variable is strongly imbalanced, the category that is the most represented will be the one generated in the majority of cases by a generative model. This needs to be addressed when choosing the parameters of our model to avoid having only one category for some variables that is represented (Jamotton & Hainaut, 2023).

For example, the variables *Customer Type*, *Type of Travel* and *Class* all have an underrepresented category. Thus, when performing clustering, it is more likely that most clusters will have values for the variable *Class* as either Business or Eco class. For the customer type, loyal customers are more dominant and for travel type, the Business type is more common. Also, we observe that the distribution between satisfied and dissatisfied within the modalities of these two variables is unequal. People traveling for personal reasons are less satisfied than customers traveling for professional purposes. Additionally, the

Business class seems to be more satisfied in comparison to the Eco and the Eco Plus classes.

Variable	Mean	Variable	Mean
Inflight wifi service	2.73	Ease of online booking	2.76
Gate location	2.98	Departure/Arrival time convenient	3.06
Food and drink	3.2	Online boarding	3.25
Cleanliness	3.29	Checkin service	3.31
Leg room service	3.35	Inflight entertainment	3.36
On-board service	3.38	Seat comfort	3.44
Baggage handling	3.63	Inflight service	3.64

Table 1: Mean values of the service-related categorical variables

The majority of service-related variables, such as *Gate location* and *Food and drink*, seem to be similarly distributed. More precisely, lower ratings and rating 5 are less represented, and a peak arises around rating 4. However, this is not the case for *Inflight wifi service*, *Ease of online booking* and *Gate location* that peak around 3, which is reflected in their mean values that are under 3 as shown in Table 1.

Conversely, *Inflight service* and *Baggage handling* show notably fewer ratings on the lower end of the scale. This is also captured in their mean values, which are located around 3.6, making them the two variables with the highest means.

Regarding the distribution of *satisfaction* across the modalities, we see a general trend where higher ratings indicate more satisfied passengers. This is quite self-evident.

	mean	sd	min	max
Departure delay	15	38,1	0	1592
Arrival delay	14,7	38,5	0	1584
Flight distance	1190,3	997,45	31	4983
Age	39,4	15,2	7	85

Table 2: Descriptive statistics of continuous variables

As a final point, we will discuss the continuous variables *Age*, *Flight distance*, *Arrival delay in minutes* and *Departure delay in minutes*. The last three exhibit a left-skewed

pattern, suggesting more of a log-normal distribution. This reflects from the barplots but also from their mean value that are rather close to their respective minimum value (see Table 2). As for the *Age*, the variable shows a bimodal distribution with peaks around 27 and 40 years old.

Regarding *satisfaction*, we observe that the delay doesn't seem to impact it significantly, as the proportion of satisfied and dissatisfied passengers stays consistent even when the delay increases. Conversely, the flight distance seems to play a role in passengers being satisfied. The longer the flight, the more satisfied people tend to be. Similarly, for the variable *Age*, passengers aged 40 to 60 tend to be more satisfied.

2.1.2 Correlation and Cramer's V matrices

By taking a brief look at the correlation matrix (see Figure 1), we get the same insights as with the barplots. Once more, *Age* and *Flight Distance* show a low, but still positive correlation with *satisfaction*. Also, we see that the departure and the arrival delay are strongly correlated with one another, which is evident.

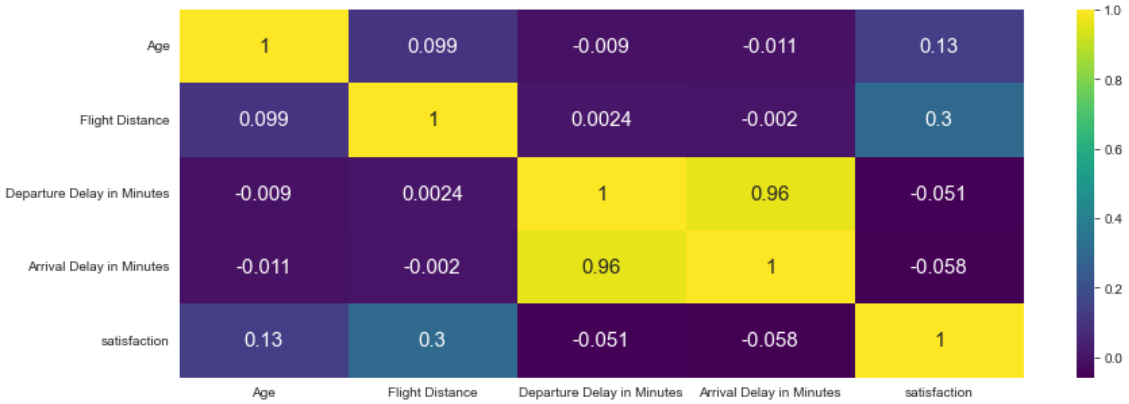


Figure 1: Correlation matrix of continuous variables with the variable satisfaction

For the categorical variables, we can examine the Cramer's V matrix from Figure 2, which shows the association between two categorical variables. The matrix can be found on the next page. The closer the value is to 1, the more strongly two variables are related. We see that *Gender* and *Departure/Arrival time convenient* have a really low association with *satisfaction*. Conversely, *Type of Travel*, *Class*, *Inflight wifi service* and *Online boarding* have the highest associations with the variable *satisfaction* ranging from 0.45 to 0.62. The other variables show associations between 0.15 and 0.42, which is moderate.

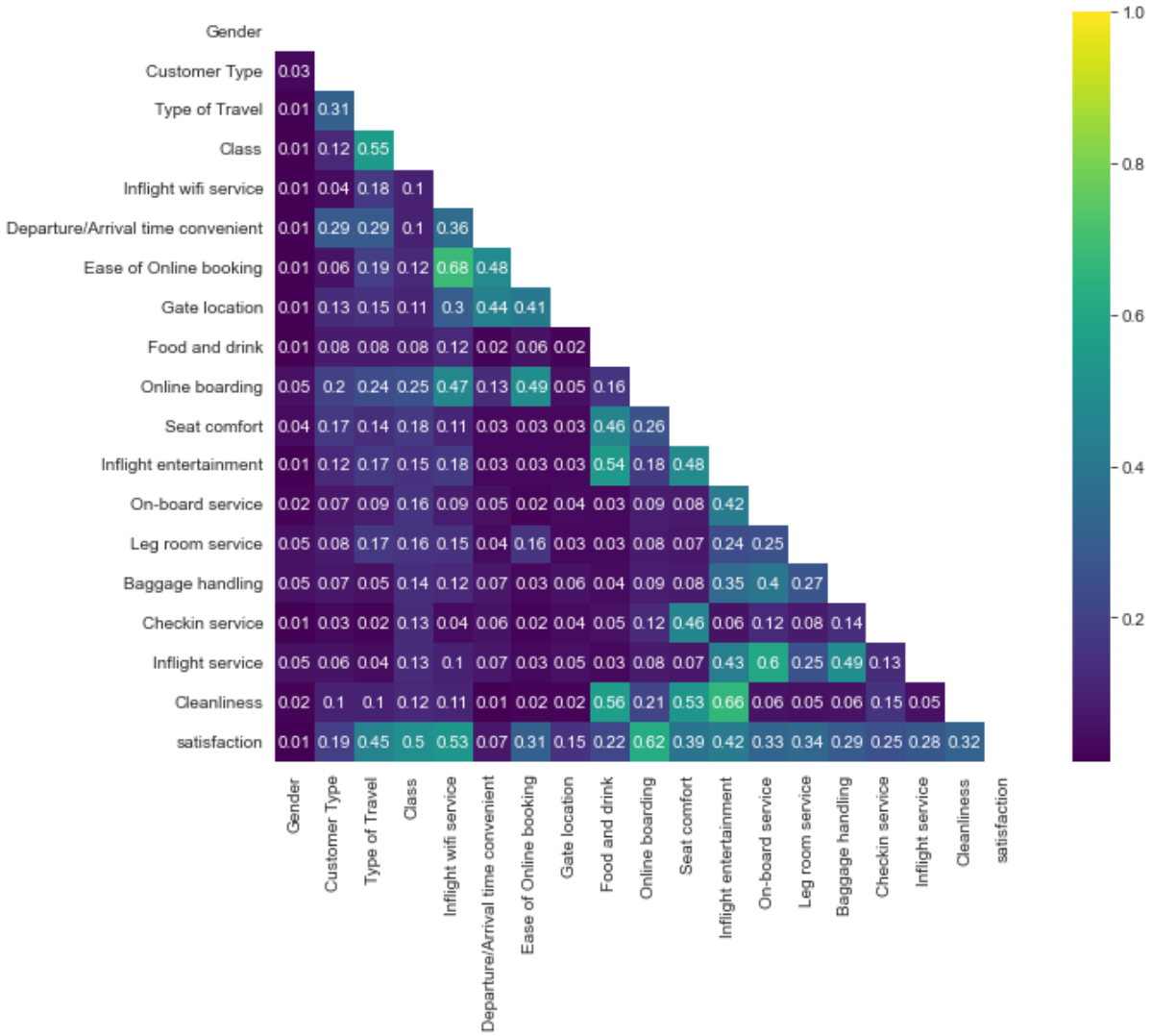


Figure 2: Cramer's V matrix

2.2 Data preprocessing

This part will go through all the preprocessing steps that were followed to prepare the data for training the models.
First things first, when downloaded from Kaggle, the data was divided into two sets : a training and a testing set. Since we will work with a k-fold cross-validation for some analyses, it is not necessary to keep those two sets separated. We chose to merge them and work with a full dataset of 129,880 observations.

2.2.1 Missing values and outliers

The only variable presenting missing values was *Arrival Delay in Minutes*, with a total of 393 occurrences. As we don't know if those are instances of planes that never landed or simply omissions, we have chosen to replace the empty values with the median value of the variable, which is 0.0 minutes.

Regarding the outliers, there didn't seem to be any aberrant values when exploring the data. The most concerning variables would be *Flight Distance* and the delays. However, a distance of 4,980 km and a delay of 1,590 minutes (approximately 26.5 hours) seem to be totally acceptable values. Thus, we won't need to deal with outliers.

2.2.2 Encoding

Multiple solutions are available for encoding the data for training a model. As we will be working with neural networks, the type of data will impact the reconstruction function to use. For example, if we have continuous variables, we can work with the mean squared error. However, we have a majority of categorical variables.

It is possible to work with a mixed encoding with NNs. Jamotton & Hainaut (2023) worked with this approach for training variational autoencoders. Each datatype is encoded separately using one-hot encoding for the categorical variables, and a quantile transformation and min-max scaling for the continuous variables for example. This results in the need of a combined reconstruction loss function using both the mean squared error and the categorical/binary cross-entropy. However, this approach won't be explored in this thesis. This leaves some room for further research.

Instead, we will categorize the continuous variables. This way, we can use the one-hot encoding on the entire dataset and treat all variables as categorical. Table 3 shows the four continuous variables that have been split up into categories.

Variable	Continuous scale	New name variable	New categories
Age	[7; 85]	AgeCat	[7; 18], [19; 30], [31; 40], [41; 50], [51; 64], [65+]
Departure Delay in Minutes	[0; 1590]	DepartDelayCat	[0; 1000], [1001; 2000], [2001; 3000], [3001; 4000], [4001+]
Arrival Delay in Minutes	[0; 15840]	ArrivalDelayCat	[0; 5], [6; 60], [61; 120], [121; 240], [240+]
Flight Distance	[31; 4980]	FlightDistanceCat	[0; 5], [6; 60], [61; 120], [121; 240], [240+]

Table 3: Recategorization of the continuous variables

After this recategorization, we will proceed to one-hot encoding. This consists of creating dummy variables for all the categories of each variable without leaving a default level. This means that we create a new column for each level of each variable and encode 1 if the subject belongs to that category, 0 otherwise. For example, a subject who is 25 years old would have their age encoded as follows :

AgeCat_7-18	AgeCat_19-30	AgeCat_31-40	AgeCat_41-50	AgeCat_51-64	AgeCat_65+
0	1	0	0	0	0

Table 4: Age encoding of a 25 years old

However, the one-hot encoding method presents some drawbacks that are important to bear in mind. As Guo & Berkhahn (2016) state in their paper on encoding techniques, the method we have chosen presents two problems. Firstly, structured data is not always continuous, as we have categorical and ordinal variables. However, neural networks are designed for continuous data and therefore perform better on such data. Although one-hot encoding offers a solution to this problem as it allows to replace categories with numbers, a second issue arises. Indeed, this encoding ignores any dependence that may exist among the levels within the same variable. Also, one-hot encoding results in a large number of variables. In our case, we initially had 23 variables and end up with 114 variables after encoding, which significantly increases the number of dimensions to work with.

2.2.3 Burt matrix and the χ^2 distance

For the clustering, we will compare the encoding using the variational autoencoder with a second technique : the Burt matrix, which measures the dependency between variables.

To create the new matrix, as explained by Hainaut (2018), the first step is to transform the dataset into a disjunctive table called the matrix D. This is equivalent to one-hot encoding, thus the matrix contains one column for each modality of each variable. The matrix D is further multiplied by its transpose in order to create a contingency table which is called the Burt matrix :

$$B = D^{\mathrm{T}} D.$$

The elements of this table represent the number of individuals that share two modalities. It is a symmetric matrix composed of K*K blocks, corresponding to the K variables of the initial dataset. Table 5 shows the contingency table of the two first variables *Gender* and *Customer Type*.

	Gender_Female	Gender_Male	Customer_Loyal	Customer_Disloyal
Gender_Female	65899	0	53056	12843
Gender_Male	0	63981	53044	10937
Customer_Loyal	53056	53044	106100	0
Customer_Disloyal	12843	10937	0	23780

Table 5: Contingency table between *Gender* and *Customer Type*

This table contains two types of blocks. First, there are the blocks $B_{i,j}$ (in white in the table) between the variables i and j (where i $\neq$ j). They represent the number of elements that share two modalities from two different variables. For example 53,056 customers are loyal customers and females. The sum of the block is the total of elements in the dataset. The second type of block is $B_{i,i}$ (in gray in the table), a block where the variable i is crossed with itself. The diagonal represents the number of occurrences in each modality of each variable. For example, the dataset contains 65,899 females and 63,981 males. The other cells are filled with 0's since it is not possible to be a member of the two modalities : one can not be a loyal and a disloyal customer at the same time. The sum of the diagonals within each group of variables is equal to the total of observations in the dataset (65,899 + 63,981 = 106,100 + 23,780 = 129,880).

The next step is to compute a distance with those contingencies which will allow us to capture the dependency between modalities (Hainaut, 2018). To do so, we use the χ^2 distance to calculate the dependence between categorical variables. Let's define the distances $\chi^2(i, i')$ and $\chi^2(j, j')$:

$$\chi^2(i, i') = \sum_{j=1}^{m} \frac{n}{n_{.,j}} \left(\frac{n_{i,j}}{n_{i,.}} - \frac{n_{i',j}}{n_{i',.}} \right)^2 \quad i, i' \in \{1, ..., m\}.$$

This represents the distance between two rows (i, i') in the Burt matrix. The second distance is equivalent but for the columns (j, j'):

$$\chi^2(j, j') = \sum_{j=1}^{m} \frac{n}{n_{i,.}} \left(\frac{n_{i,j}}{n_{.,j}} - \frac{n_{i,j'}}{n_{.,j'}} \right)^2 \quad j, j' \in \{1, ..., m\}.$$

Now, as the Euclidean distance is preferred for distances, we replace $n_{i,j}$ by a weighted version of it. Also, we note the following links : $n_{i,.} = Kn_{i,i}$ and $n_{.,j} = Kn_{j,j}$. For $n_{i,.}$, this implies that summing over all the columns for row i is equivalent to taking the sum of the row i in the first block and multiplying by the number of variables. In our earlier example, if we want the sum over row 1, it is equivalent taking $65,899 + 53,056 + 12,843$ or to multiply the sum from the first block by the number of blocks, thus $65,8992$. This is also true for the columns. We end up with the weighted $n_{i,j}$:

$$n^W_{i,j} := \frac{n_{i,j}}{K\sqrt{n_{i,i}n_{j,j}}} \quad i, j = 1, ..., m.$$

Using the diagonal matrix C = $diag(n_{11}^{-\frac{1}{2}}...n_{mm}^{-\frac{1}{2}})$, B^W the weighted Burt matrix is given by $B^W = \frac{1}{K}CBC$ and $\chi^2(i, i')$ and $\chi^2(j, j')$ are also rewritten in terms of the weighted row and column sums :

$$\chi^2(i, i') = \sum_{j=1}^{m_2} \left(n^W_{i,j} - n^W_{i',j} \right)^2, \chi^2(j, j') = \sum_{i=1}^{m_1} \left(n^W_{i,j} - n^W_{i,j'} \right)^2.$$

For encoding the data, the final step is to multiply this weighted Burt matrix with the initial dataset that was one-hot encoded and divide the product of those matrices by the dimension of the initial dataset denoted by l. In our case, the dimension is 22. The formula for this final set is $X_{Burt\_encoded} = DB^W/l$.

3 Variational Autoencoders

3.1 Neural Networks

As a preliminary step, we will define the concept of neural networks (NNs) as the variational autoencoders are based on them. This part is inspired by the book of Zaki & Meira, *Data Mining and Machine Learning : Fundamental Concepts on Learning Representations* (2020).

Neural networks are inspired by the human brain and its neuronal networks, where neurons send information to other neurons via dendrites when the concentration of ions is sufficient for activation. They are a fundamental part of machine learning and are used for various tasks such as regression and classification.

Similarly to the biological neuronal networks, a neural network is composed of nodes called neurons that are interconnected and organized in layers, starting from the input layer and extending to the output layer. The layers in between are called hidden layers. Each neuron receives information from neurons of the previous layer and processes it accordingly to an activation function. This function helps to determine if the neuron should be activated and forward information to the next layer. Neurons between each layer are connected by links, which carry weights. These weights determine the activation of a neuron and the importance of the input from a previous neuron to the evaluated neuron. These weights, as well as biases, are updated through forward and backpropagation.

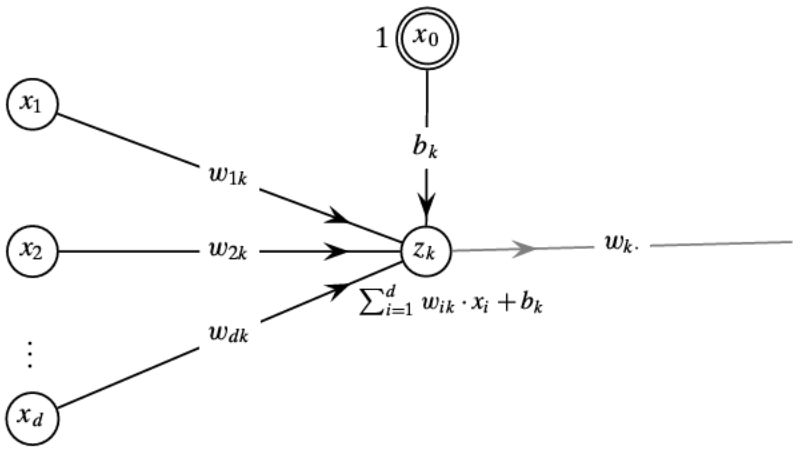


Figure 3: Artificial neuron (source : Zaki & Meira, 2020)

x_0 is a special node, a bias node, whose value is always equal to 1. Its input value on z_k will be defined by the bias coefficient b_k. The activation function will thus activate a neuron if the sum of the inputs, $\sum_{i=1}^{d} w_{ik} * x_i + b_k$ (see Figure 3), is higher than a certain

threshold depending on the chosen activation.

Some existing activation functions are :

- The linear/identity function which simply returns the value of the node.
- The rectified linear unit (ReLUs) function, where the neuron stays inactive if the input is lower or equal to zero and above that threshold, it works the same way as the linear function;
- The sigmoid function first transforms the input value with the function $f(net_k) = 1/(1+\exp\{-net_k\})$ and the output is passed through when this value is bigger than 0.5.

Other possibilities include the hyperbolic tangent activation function, the SeLU, the softplus, and so on.

All neurons from one particular layer can have different activation functions, but here we will work with the same activation function. Also, if the network has multiple hidden layers, each hidden layer can have its own activation function. Bias neurons and the input layer use the identity function, thus sending the full value of the neuron to the next neuron.

In regression tasks, we will prefer the identity function. For classification problems, a good strategy is to work with the sigmoid or the softmax (not explained here) functions on the last layer, as their output values are between 0 and 1. For the hidden layers, the sigmoid, tanh, SeLU or ReLU are good choices.

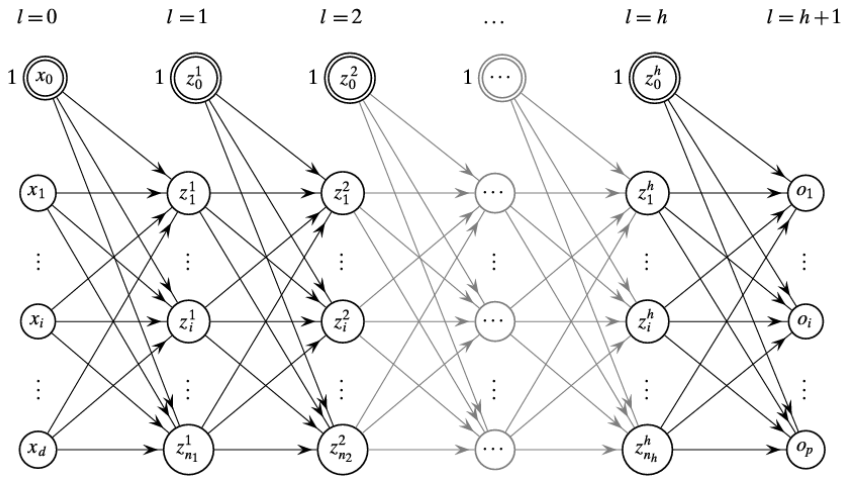


Figure 4: MLP with h hidden layers (source : Zaki & Meira, 2020)

A common type of neural networks is the multilayer perceptron (MLP), with one or multiple hidden layers, then called deep neural networks. See an illustration of such a model in Figure 4. They are feed-forward networks and have fully connected layers, meaning that all neurons from one layer are connected to the neurons of the next layer. Also, feed-forward means that information flows from one layer to another, starting from the input layer and ending in the output layer.

Learning occurs through feed-forward and backpropagation. The weights and biases are first initialized, then the feed-forward phase serves to compute the outputs values going through the entire network layer by layer. The error is then computed, and the weights are learned by backpropagation, meaning that the error gradient will be propagated from the output to the input layer, layer by layer. For each node, we compute the extent to which it deviates from the actual value and update the weights proportionally to the error. A large error leads to a big change in the weight. This is first done to the weight between the output and the before last layer, and it is repeated to each pair of layers sequentially.

To calculate the error during the feed-forward phase, different functions can be used depending on the type of dataset used. Some regression losses are the mean squared error, mean absolute error or the mean absolute percentage error. For classification problems, we prefer using the entropy. For example, we can use the binary cross-entropy or the categorical cross-entropy as well as the Kullback-Leibler divergence or the Poisson log-likelihood.

The update of the weights and biases can be done via gradient descent, where we update the weights or biases by taking a small step (denoted by η, also called the learning rate) towards the opposite direction of the gradient. Let's consider w_{ij}^l as the weight between node i from hidden layer l (z_i^l), the node j of layer $l+1$ (z_{ij}^{l+1}) and b_j^l the bias term between those two layers. The updated weight and bias are given by :

$$\begin{aligned} w_{ij}^l &= w_{ij}^l - \eta * \nabla_{w_{ij}^l} \\ b_j^l &= b_j^l - \eta * \nabla_{b_j^l} \end{aligned} \tag{1}$$

Computing the error and updating the weights and biases is done repeatedly for a number of iterations. Each iteration is called an epoch, which is a hyperparameter of the model that need to be chosen before training.

Computing the gradient for the entire dataset at once would be fastidious and time-consuming. An alternative is stochastic gradient descent, where the gradient is computed for a random subset of the dataset called a minibatch. The size of the minibatch is called the batch size and is, as the number of epochs, a parameter of the neural network.

SDG is a variation of the gradient descent algorithm, which considers the entire dataset at each update step. This is the most commonly used algorithm, but other options exist. Let us explain three of them :

- The adaptative gradient algorithm (AdaGrad) is often used when data is sparse. It is stochastic gradient descent with adaptative learning rates.
- Root mean square propagation (RMSprop) also updates the learning rate to each neural weight. Taking the momentum of recent gradients, we compute a moving average and divide the weight by this latter.
- The adaptative moment estimation (ADAM) is based on RMSprop, where we only take the two first moments of gradients.

The batch size, the number of epochs, the loss function, the optimization algorithm (called the optimizer), and the learning rate of the optimizer are all hyperparameters of a neural network that will have an impact on the results and performance of a model. Also, the number of hidden layers, the number of neurons on each of these layers, and the activation function for each layer need to be chosen wisely. All those parameters depend on the context and the data we are working with.

3.2 Autoencoders

3.2.1 The model

As explained by Li et al. (2023), an autoencoder is a particular type of neural network that is trained to reproduce its input while minimizing the reconstruction error. It is therefore composed of two models each constituted of at least two layers. Figure 5 illustrates the architecture of the simplest autoencoder.

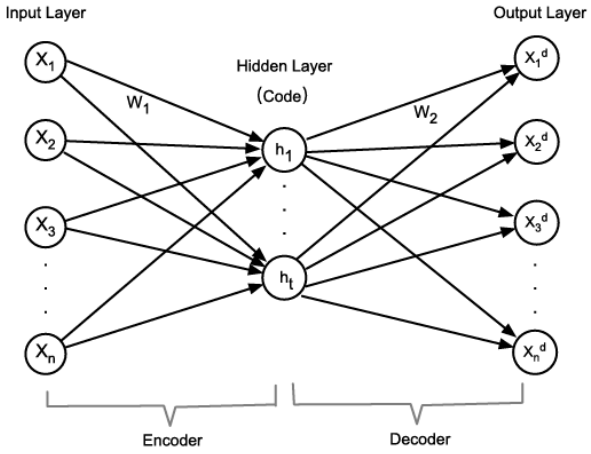


Figure 5: Architecture of a simple autoencoder (source : Li et al., 2023)

The first model is the encoder, which aims at compressing and encoding the data into fewer variables, thereby reducing the dimensionality. In the simplest configuration, this model is formed by an input and a hidden layer. There is an input X that is passed through the encoder, which converts it into a mapping h. This transformation occurs based on the activation function outlined below, where W_1 is the weights matrix and b_1 the bias vector.

$$h = f(x) = f(W_1 X + b_1)$$

The second model is the decoder, whose purpose is to restore the compressed data in order to reconstruct X^d, the reconstructed input. It is again made up of an input, a hidden and an output layer. The formula representing the process is the following :

$$X^d = g(x) = g(W_2 h + b_2)$$

$f(x)$ and $g(x)$ are activation functions. While the encoder activation function needs to be a non-linear activation function, such as a sigmoid or tanh function, the decoder activation function can also be an affine transformation function. This is a linear transformation

followed by a translation of the data or even a simple linear transformation making it linear autoencoders (Bank et al., 2023).

The complete model is trained to find the weights and the bias vectors (represented by the parameters vector θ) that minimize the total error between the original and the restored data. In general, the number of neurons in the hidden layer is lower than in the other two layers. Also, if this is the case, a bottleneck is imposed to the autoencoder. If the number of neurons in the hidden layer is equivalent or higher than in the input layer, then we need to find a way to enforce the encoder to learn a different function than the identity transformation (Bank et al., 2023).

We discussed the need to minimize the reconstruction error. The two general types of loss functions used in regression and classification problems respectively are the squared error and cross-entropy. In our case, it is more appropriate to work with cross-entropy. Here is its loss function ($J_{AE}(\theta)$) :

$$J_{AE}(\theta) = J(X, X^d) = -\sum_{i=1}^{n}(x_i log(x_i^d) + (1 - x_i) log(1 - x_i^d))$$

where X is the input data and X^d is the reconstructed data. For the optimization, we can use stochastic gradient descent (SGD) or any of the optimization algorithms introduced in the previous section.

Autoencoders work great for feature extraction while most of the time avoiding overfitting. However, because the model needs to be trained layer by layer, it takes more time than working with a more conventional method. Also, the interpretability of the model is more complicated than linear regression for example, and the accuracy is sometimes insufficient (Li et al., 2023).

Autoencoders are also widely used for their ability to reduce dimensionality, which helps for representation purposes. Autoencoders are actually equivalent to PCA if all non-linear operations are removed. More precisely, they are a generalization of PCA (Bank et al., 2023).

3.2.2 Regularization and variants

The most common form of autoencoder is the one presenting a bottleneck. This solution helps reduce dimensionality and extract helpful features. However, if the number of neurons in the hidden layer is greater than or equal to the input layer, or if the bottleneck results in overfitting, we may use some regularization techniques.

The Regularization Autoencoder (ReAE) was introduced to avoid overfitting and control the degree of weight reduction. In that model, we add a regularization term to the chosen loss function :

$$J_{ReAE} = J(X, X^d) + \lambda ||W||_2^2$$

Other possible regularizations presented by Li et al. (2023) are as follows :

- Denoising Autoencoder : In this version of the autoencoder, noise is added to the input data after the input layer to reconstruct a pure version of the input. The output of this model is thus robust to the noise in the input. When the autoencoder is trained gradually layer by layer, it is referred to as a stacked DAE, which is a network of multiple DAEs. In this model, the output from the previous layer is used as clean input to which noise is added each time before training. Adding too much noise can lead to underperformance of the model, and this process takes more time.

- Sparse Autoencoder : This method can be used either as a replacement for the bottleneck or in addition to it to reduce overfitting. The goal is to add a sparcity constraint on the hidden layer. This can be done either with L_1 regularization or by using the KL-divergence. A faster version was introduced, the k-sparse encoders. Instead of using regularization functions, linear transformation is used to compute the activation values, and the k-largest values are kept. However, it was not found yet how to define the optimal k.

- Contractive Autoencoder : The purpose of this upgrade of the autoencoder is to make the feature extraction process robust against small perturbations by ignoring small changes in the input that do not have a significant impact on the reconstruction. To achieve this, we add a penalty term to the Jacobian matrix of the conventional autoencoder. Smaller variations in the latent space are thus lessened by the regularization factor, allowing the larger variations to remain.

- Variational Autoencoder : They are used to generate data through a probabilistic distribution. More on this model is to be found in the next section.

- Many more variants of the conventional autoencoder exist such as convolutional AEs, marginalized denoising AEs, kernel method-based AEs, and so on.

3.2.3 Applications of Autoencoders

Autoencoders serve for many unsupervised and supervised tasks, such as classification, clustering, anomaly detection, and dimensionality reduction. They can be applied to diverse types of data, including images, textual data, and speech data, among others. In

this section, we go through some of the possible applications presented by Bank et al. (2023).

A first application is anomaly detection. This consists of learning the characteristics of a normal profile, of an insured for example, using a model by only feeding it with normal profiles. Then, the model should recognize anomalies when confronted to data deviating from the norm. An anomaly passing through the autoencoder would result in a larger reconstruction error compared to a normal sample.

Autoencoders can also be used for classification when the proportion of labeled data is small. This is a semi-supervised task. The autoencoder is fully trained, but only the encoder is used as feature extractor and combined with a classification network. The intuition behind is that data that are similar are expected to have a similar representation in the latent space when compressed using the autoencoder.

Furthermore, many supervised and unsupervised models have to deal with the curse of dimensionality when working with high-dimensional data. There exist various dimensionality reduction approaches such as Principal Component Analysis (PCA), and Linear Discriminant Analysis (LDA). Compared to PCA, autoencoder is a non-linear technique and is therefore often able to achieve higher performance in reconstructing the input.

Clustering is also sensitive to dimensions and suffers from the curse of dimensionality. Autoencoders reduce the dimensionality of the data while extracting meaningful features. As Song et al. showed in 2013, using an autoencoder to reduce the dimensionality and then proceeding to clustering using the K-means algorithm greatly increases performance compared to using the original space. Furthermore, they introduced a version of an autoencoder, Clustering Based on Autoencoder, which includes cluster assignment in the objective function. By doing this, clusters are updated iteratively to ensure that data points in the latent space are close to their cluster center while keeping an eye on the reconstruction error. We won't try to reproduce this approach with variational autoencoders in this thesis, but this could be part of future research.

Finally, autoencoders can be used as a generative model in their variational form. The trained variational autoencoder is able to generate new meaningful data by feeding the decoder with samples of random variables drawn from a given prior.

3.3 Variational autoencoders

3.3.1 Introduction

After explaining what a neural network is and building upon that to define the autoencoder, we can dive deeper into the variational autoencoders, which are a Bayesian extension of AEs. This model was proposed by Kingma et al. in 2013.

With autoencoders, each input point x_i is encoded independently of the rest of the input. As a result, the latent space is likely to be discontinuous. This poses a problem for data generation, as adding a variation to a latent data point would probably lead to a gap in the latent space. However, since the model was not trained on that specific point, the decoder may not know how to handle it. Continuity and completeness are necessary for a generative model to work, but the conventional autoencoder comes short to those two points. To solve this problem, a Bayesian extension is added to the basic autoencoder, allowing it to learn a probabilistic latent representation of the input instead of a deterministic one.

3.3.2 The objective function

As mentioned above, our objective is to reconstruct an input x by first using an encoder to map the data into a latent space and then passing it through a decoder to reconstruct the input as accurately as possible. Therefore, for each data point, we need to determine the latent variables z that allow to produce an output very similar to the input x.

More formally, we have a sample of z drawn from a probability density function $p(\mathbf{z}) = N(\mathbf{z}|0, I)$, this is a prior distribution over $\mathbf{z}$ following an isotropic normal distribution. In order to reconstruct x, we use a deterministic function $f(\mathbf{z}; \theta)$, where θ, the set of parameters, is unknown. Our goal is to optimize θ such that the probability of reconstructing x using $f(\mathbf{z}; \theta)$ as similarly as possible to the original x high when sampling $\mathbf{z}$ from $p(\mathbf{z})$. To make the dependence between x and z more explicit, we replace $f(\mathbf{z}; \theta)$ with $p_\theta(\mathbf{x}|\mathbf{z})$, which represents the conditional distribution of $\mathbf{x}$ given $\mathbf{z}$ and the parameter vector θ.

The standard criterion used when working with probabilistic models is the maximum log-likelihood. To compute this, we need to consider the marginal likelihood of x, which represents the probability of observing x when going through the full generative process. We refer to the Bayes' theorem. Indeed, we know that :

$$p_\theta(z|x)p_\theta(x) = p_\theta(x, z) = p_\theta(x|z)p_\theta(z) \tag{2}$$

The marginal likelihood of x is found by integrating this equation over the latent variables.

We then get the following formula :

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz = \int p_\theta(z|x)p_\theta(x)dz \tag{3}$$

The logarithm of this equation is then taken to obtain the marginal log-likelihood. We can then maximize this formula using an appropriate optimization method to find the set of parameters θ that will maximize the probability of observing x when going through the entire generative process.

However, there are aspects to keep in mind when solving equation 3. The first question that arises is how to define z and what distribution the latent variables should follow. VAEs propose a simple normal distribution for the latent variables, stating that normally distributed variables can be used to map any d-dimensional distribution using a complicated enough mapping function, such as a deep neural network. Thus, a first layer in the VAE would draw a random sample of ϵ from an isotropic Gaussian distribution, then map z by transforming that sample. This is called the reparametrization trick.

The second problem to consider is how to integrate over z in order to find $p(x)$. The integral is indeed intractable, and $p_\theta(z|x)$, the posterior distribution of z given x, is unknown. The VAE framework solves this problem by proposing to use an approximation of the posterior distribution, allowing to rewrite the equation 3 as a sum of the Kullback-Leibler divergence and the Variational/Evidence Lower Bound (ELBO). The derivation comes from the following demonstration proposed by Kingma and Welling (2019) in their introduction to variational autoencoders :

$$\begin{aligned}
log\ \ p_\theta(x) &= \mathbb{E}_{q_\theta(z|x)}[log\ \ p\theta(x)] \\
&= \mathbb{E}_{q_\theta(z|x)}\left[log\left[\frac{p_\theta(x,z)}{p_\theta(z|x)}\right]\right] \\
&= \mathbb{E}_{q_\theta(z|x)}\left[log\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\ \ \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]\right] \\
&= \mathbb{E}_{q_\theta(z|x)}\left[log\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\right]\right] + \mathbb{E}_{q_\theta(z|x)}\left[log\left[\frac{q_\phi(z|x)}{p_\theta(z|x)}\right]\right] \\
&= D_{KL}(q_\theta(z|x)||p_\theta(z|x)) + \mathcal{L}(x;\theta,\phi)
\end{aligned} \tag{4}$$

The left term is the KL divergence and the right term is the ELBO. Let us explain the computations behind this.

3.3.3 The inference model and the approximate posterior

First, we introduce $q_\phi(z|x)$, which is an approximate posterior or variational posterior with ϕ as the set of approximate parameters. Given some value of x, it returns a distribution over z values that are likely to produce the given x. It is a tractable approximation for the true posterior $p_\theta(z|x)$. As we want those two distributions to be as close as possible by optimizing over ϕ, we use the Kullback-Leibler divergence which is a measure of dissimilarity between two distributions :

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{Y}\sim p}[ln\ \ q(\mathbf{Y}) - ln\ \ p(\mathbf{Y})]$$

The goal is to minimize the KL divergence, which is never negative. If the divergence is close to 0, then there is almost no difference between the true and the approximate posterior.

ϕ is predicted using a neural network, which is called an inference or encoder network. The input is passed through the encoder network outputting a set of parameters : the mean μ and the logarithm of the variance σ^2, for each latent data point. The logarithm of the variances is taken to avoid learning negative values for the variance through back-propagation. In parallel, a sample of ϵ is drawn from an isotropic Gaussian prior. The variational posterior is thus taken to be an isotropic multivariate Gaussian distribution :

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(z|\mu_z(\mathbf{x}), \Sigma_z(\mathbf{x}))$$

The encoder network with any number of hidden layers can be summarized by :

$$\begin{aligned} \epsilon &\sim \mathcal{N}(0, \mathbf{I}) \\ (\mu, \log(\sigma^2)) &= EncoderNeuralNet_\phi(\mathbf{x}) \\ \mathbf{z} &= \mu + \sigma \odot \epsilon \end{aligned} \tag{5}$$

We have introduced ϵ in the equations above. More explanations on that element will follow when we talk about the reparameterization trick later on.

This inference model is an advantage of the VAE compared to the Variational Inference framework where each data-case has its own distribution. For big datasets, this would be fastidious. Instead, the VAE learns one set of parameters ϕ to transform the input data to the latent data which is called amortized inference.

3.3.4 Evidence lower bound

Now, let's return to the marginal log-likelihood. We have defined the Kullback-Leibler divergence and explained its role. The second part of the formula is the Variational or

Evidence Lower Bound (ELBO). Conversely to the KL divergence, which will be minimized, the ELBO will be maximized. With the KL divergence being non-negative, the ELBO serves as a lower bound to the log-likelihood. As shown by Kingma et al. (2019) :

$$\mathcal{L}(x;\theta,\phi) = \log p_\theta(x) - D_{KL}(q_\theta(z|x)||p_\theta(z|x)) \quad \leq \quad \log p_\theta(\mathbf{x}) \tag{6}$$

The KL divergence is thus a measure for the goodness of fit between the true and the approximate posterior but it also indicates the tightness of the bound, which is the gap between the ELBO and the marginal log-likelihood. The better the approximation of the posterior, the tighter the gap.

When optimizing the model, we will focus on the ELBO. Maximizing this latter will, on the one hand help to maximize the marginal log-likelihood, and on the other hand, it will minimize the Kullback-Leibler divergence, aiding in having an approximation of the true posterior as accurate as possible. Moreover, as the ELBO depends on both sets of parameters ϕ and θ, they will both be optimized when maximizing the ELBO.

Assuming we have a dataset $\mathcal{D}$, the objective function for the dataset is equal to the sum of the individual ELBOs for each data point. The objective function becomes $\sum_{\mathbf{x}\in\mathcal{D}} \mathcal{L}_{\theta,\phi}(\mathbf{x})$, of which we will take the gradient. However, the ELBO and its gradient are intractable. This is resolved, when the latent variables are continuous, with a reparameterization trick that allows optimizing the ELBO with stochastic gradient descent.

3.3.5 Reparameterization trick

As we just mentioned, a reparameterization trick is employed when the latent variables are continuous, and the two networks, encoder and decoder, are differentiable. If these conditions are met, a change of variable can help deriving the gradient of the ELBO with respect to the two sets of parameters ϕ and θ.

First, we transform $\mathbf{z}$ using another random variable ϵ : $\mathbf{z} = \mathbf{g}(\epsilon, \phi, \mathbf{x})$. This transformation is differentiable and ϵ has a distribution independent of $\mathbf{x}$ and ϕ. Now we can rewrite the expectation with $\mathbf{z}$ being equal to the new transformation :

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{p(\epsilon)}[f(\mathbf{z})]$$

If we take the gradient of this expectation, the gradient and the expectation are now commutative operators. Thus, we get the expectation of the gradient of $f(\mathbf{z})$ with $\mathbf{z}$

generated from the transformation and $\epsilon \sim p(\epsilon)$:

$$\begin{aligned} \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] &= \nabla_\phi \mathbb{E}_{p(\epsilon)}[f(\mathbf{z})] \\ &= \mathbb{E}_{p(\epsilon)}[\nabla_\phi f(\mathbf{z})] \\ &\simeq \nabla_\phi f(\mathbf{z}) \end{aligned} \tag{7}$$

Now when applied to the ELBO, we rewrite it with the reparameterized $\mathbf{z}$ replacing $q_\phi(\mathbf{z}|\mathbf{x})$ by $p(\epsilon)$:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{p(\epsilon)}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})]$$

With the same logic, using the commutation of the gradient and the expectation, we derive a Monte Carlo estimator of the ELBO of each observation using a unique sample ϵ drawn from $p(\epsilon)$:

$$\begin{aligned} \epsilon &\sim p(\epsilon) \\ \mathbf{z} &= \mathbf{g}(\phi, \mathbf{x}, \epsilon) \\ \bar{\mathcal{L}}_{\theta,\phi}(\mathbf{x}) &= \log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \end{aligned} \tag{8}$$

We refer to this as a Monte Carlo estimator because the marginal likelihood is computed through simulations with random samples of $\mathbf{z}^{(l)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ from the inference model :

$$\log p_\theta(\mathbf{x}) \approx \log \frac{1}{L} \sum_{l=1}^{L} p_\theta(\mathbf{x}, \mathbf{z}^{(l)}) / q_\phi(\mathbf{z}^{(l)}|\mathbf{x})$$

As the ELBO is the difference between the marginal likelihood and the KL divergence, the ELBO is also computed through simulations. The larger L, the closer the estimation will be to the actual marginal likelihood. However, as we have many observations, setting $L = 1$ is often sufficient.

Once we have the Monte Carlo estimator of the ELBO, we can compute its gradient and start the optimization using SDG, Adam, RMSprop, or any other optimization technique.

To summarize this section, the architecture described is visually represented in Figure 6. We see the input that is passed through the encoder. Then a random ϵ is sampled and used to generate the latent variables z. The encoded data again traverses through a MLP, this time the decoder, in order to reconstruct the input.

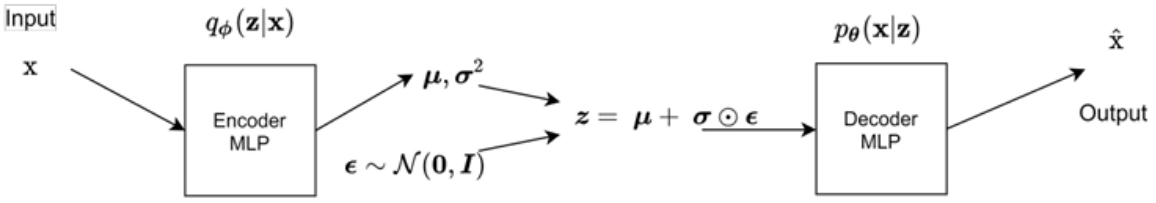


Figure 6: Variational Autoencoder (source : Singh & Ogunfunmi, 2021)

3.3.6 Some challenges

Over the years, some weaknesses have appeared in the so-called vanilla VAE. These have been addressed in research, and we will discuss a few of them in what follows. We will also present some of the variants that were introduced to remedy these challenges.

A first problem is that the objective sometimes gets stuck during the optimization process. Bowman et al. (2015) and Sønderby et al. (2016a) proposed to anneal the latent cost (KL divergence) over the epochs, while Kingma et al. (2016) introduced the method of free bits, where each latent variable needs to encode a minimum amount of information.

Some blurriness in the output images was also observed due to variance loss from the input to the output. This results in some values of $(\mathbf{x}, \mathbf{z})$ being likely under the inference model but not under the generative model. To solve this problem, flexibility can be added to the inference or generative model by introducing auxiliary latent variables or working with Normalizing Flows (Rezende & Mohamed, 2015), Inverse Autoregressive Transformations (Kingma et al., 2016), among others.

Furthermore, the VAE deals with a balancing issue in the context of images. The loss function is indeed composed of two elements : the KL divergence and the reconstruction loss. The first term aims at regularizing the latent space, while the other controls the quality of the output. By emphasizing one over the other, either the generation of new samples will be of better quality, or the disentanglement and the reconstructed outputs will be better. A balancing factor is introduced in the 2-Stage VAE to balance these effects (Singh & Ogunfunmi, 2021).

The last challenge, which we have briefly touched upon, is disentanglement. This is the principal purpose of dimensionality reduction techniques. This defines the ability of a model to capture the variation into the latent space in order to have latent variables be all independent of each other. Thus each latent variable would represent one variation of the data. With their bottleneck, VAEs offer this possibility. However, vanilla VAEs don't succeed in presenting the best disentangled representations. Other variants such as

the β-VAE or the INFOVAE have tried to improve the representation of the latent space (Singh & Ogunfunmi, 2021).

3.3.7 Variants

Over the years and through research, variants of variational autoencoders have been proposed addressing the challenges that have arisen (Wei et al., 2020; Singh & Ogunfunmi, 2021). Here are some of those variants :

- β-VAE : The goal of this variant is to improve disentanglement by adding a factor β that allows for the reweighting of the effect of the KL divergence in the loss function (Higgins et al., 2022).
- INFOVAE : Also known as MMD-VAE, this variant aims at improving representation learning by scaling the divergence with a regularization term. Moreover, the mutual information between x and z is added to the loss and rescaled by a factor α. However, neither this technique, nor the β-VAE solve the problem of blurriness (Zhao et al., 2018).
- VQ-VAE : Unlike other versions of the conventional VAE, this adaptation tries to learn discrete latent variables instead of continuous ones. This is particularly useful for complex reasoning and planning where discrete representations are needed. The model uses Vector Quantization (VQ) which groups points located in an area into one latent variable. However, it is less stable when training on challenging datasets (van den Oord et al., 2017).
- Variational Deep Embedding (VaDE) : This technique performs well for clustering and data generation as it works with a Gaussian Mixture Model in the latent space for clustering purposes (Jian et al., 2016).
- Gaussian Mixture VAE (GMVAE) : Similar to the VaDE, a GMM is introduced, but this time as the prior. It is also effective for performing clustering, although the computational complexity is higher compared to other deep clustering algorithms (Dilokthanakul et al., 2016).
- VAE-GAN : This is a combination of VAE and GAN with the purpose of improving the quality of outputs. The structure is the same as a VAE, but with a GAN discriminator after the decoder network. However, it is computationally expensive and less stable during longer trainings (Larsen et al., 2015).

3.3.8 Applications of Variational Autoencoders

VAEs share some application tasks with the conventional autoencoders, such as data representation, classification in a supervised and semi-supervised layout, as well as new applications such as data generation. Semi-supervised tasks have shown great results as a consequence of the generative model of the variational autoencoder, as presented in the papers of Rezende et al. (2016) and Pu et al. (2016). Data generation is also an interesting application that has been tested and used in many fields such as astronomy, chemistry, and more.

Variational autoencoders are mostly used and known for image recognition and generation, as well as speech/audio and text data. These are also the applications of the VAEs that are the most researched.
In 2021, Singh and Ogunfunmi explored an unresearched usage of this technique in their paper about the analysis of signal processing or times series. They applied the model to financial data as well as speech source separation tasks and bio-signals.

In this thesis, VAEs will be applied on tabular data as this is still a relatively unexplored field. Salim A. (2018) has investigated in their paper the possibilities of usage of the variational autoencoder in the medical domain in order to generate patient data. We see that the new records are well mixed in the actual patient records when performing a principal component analysis after training the VAE. Additionally, Jamotton & Hainaut (2023) have explored the generation of synthetic data in the field of insurance showing how new data records maintain the properties of the original data.

Clustering, classification, dimensionality reduction and data generation are the applications that will interest us the most in this thesis. As explained earlier, we will first implement a variational autoencoder to reduce the dimensionality of the dataset and extract meaningful features. With that latent data, we will be able to proceed to clustering with the K-means algorithm, classification using a generalized linear model (GLM) and finally represent the data in a low-dimensional space. In a second phase, we will generate data with the decoder network and perform clustering on the newly generated observations.

4 Additional techniques

As explained in the introduction, we will perform dimensionality reduction using the variational autoencoder. With the obtained encoded data, we can perform various analyses such as clustering, regression, and so on.

In this thesis, we will concentrate on three tasks :

- visualization using t-SNE;
- unsupervised clustering with the K-means algorithm;
- regression in order to predict the probability that a passenger would be satisfied or not using a generalized linear model.

In this section, we thus briefly present these three algorithms.

4.1 K-means

This section will explain the K-means algorithm based on Zaki and Meira's book published in 2020. The K-means algorithm is a clustering algorithm that aims to find k groups or clusters of close data points. The cluster C_i is then represented by a centroid which is the mean of the n_i points in the cluster :

$$\mu_i = \frac{1}{n_i} \sum_{x_j \in C_i} \mathbf{x}_j$$

In order to find the optimal clustering, the algorithm uses the sum of squared errors scoring function which is the sum of distances between each point and the centroid of the cluster it belongs to :

$$SSE(\mathcal{C}) = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||\mathbf{x}_j - \mu_i||^2$$

The optimal clustering is the one minimizing the SSE. This means that we aim to minimize the sum of the distance of each data point to its cluster center.

First, the number of clusters k is chosen, and k centroids are randomly initialized. Then, data points are assigned to the closest centroid, and the centroids are updated with the new data points. These two steps, re-assignment and updating of the centroids, are repeated until no further improvement can be made, meaning that the centroids do not change from one iteration to the next. The algorithm needs to be run multiple times from

scratch because the initialization of the centroids is random, and it could converge to a local minimum. At the end of the runs, the clustering with the lowest sum of squared errors is chosen.

4.2 Generalized Linear Models

Remember the classical linear model (OLS) :

$$Y = X\alpha + e, \quad e \sim_{iid} N(0, \sigma^2)$$

which can also be written in terms of μ, the mean response, and e, the residual term :

$$Y = \mu + e, \quad \mu = E[Y] = X\alpha$$

A generalized linear model is a supervised learning technique that aims to predict a response value Y given some explanatory variables X through regression. It is a generalization of the OLS with two adaptations (Legrand, 2021) :

1. The target variable is not necessarily normally distributed. Instead, it should belong to the exponential family of distributions.

2. The link between the explanatory variables and the mean response is not necessarily linear; it is mapped by a link function to scale the linear predictor (the sum of the products of the explanatory variables and their coefficients) into the domain of the response. For example, the domain of a Bernoulli response, $Y \sim Bern$, is $]0; 1[$. The relation using the link function $g()$ for the ith observation is of the form :

   $$g(\mu_i) = \mathbf{x}_i^T \beta, \quad \mu_i = E[Y_i]$$

   where $\mathbf{x}_i^T$ is the ith row of the design matrix $\mathbf{X}$, and β is a parameter vector to be estimated.

A generalized linear model comprises three elements :

- the response distribution, which belongs to the exponential family of distributions;
- the linear score, which corresponds to the linear combination of the coefficients and the explanatory variables : $score = X\beta$;
- the link function, as mentioned above, which maps the score to the mean value.

We will go through each component.

4.2.1 Exponential family

First, we mentioned that Y needs to belong to the exponential family. The exponential family of distributions gathers all distributions that can be expressed in terms of the known functions a, b and c :

$$f(y;\theta;\phi) = \exp\left[\frac{y\theta - b(\theta)}{a(\phi)} + c(\theta,\phi)\right].$$

Here, θ is the location parameter and ϕ defines the dispersion in distributions that utilize two parameters (e.g., the gamma and the normal distribution).
The Poisson, normal, gamma, binomial and Bernoulli distributions are part of this family. We will only focus here on the Bernoulli distribution, which is suitable for our data.

The response Y represents the success or failure for an event, with values bounded between 0 and 1, denoting the probability of success. The distribution of Y if in the form $Y \sim Bernoulli(\pi)$, where π is the probability of success. The distribution is :

$$f(y) = \pi^y(1-\pi)^{1-y}, \qquad y = 0, 1$$

The exponential form is given by :

$$f(y;\theta) = exp\big[y\theta - \log(1+\exp(\theta)) + 0\big]$$

with $a(\theta) = 1$, $b(\theta) = \log(1 + exp(\theta))$, and $c(y) = 0$. This distribution doesn't have a dispersion parameter.

Other interesting characteristics of this distribution include the mean $E(y) = \mu = \pi$ and the variance $V(y) = \mu(1-\mu)$.

4.2.2 Linear Score

The second component of the GLM is the linear score. For a response Y_i, the score is defined as :

$$score_i = \mathbf{x_i}^T\beta = \beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}, i = 1, 2, ..., n.$$

Here, the β_j's are the unknown regression coefficients that need to be estimated. β_0 is known as the intercept, representing the common term to all scores when all features x_{ij} are set to 0. Each β_j measures the influence of an additional units in x_{ij} (Denuit et al., 2019a). For example, if β_1 has a coefficient of 1.3, adding one unit of x_{i1} increases the score by 1.3.

X is referred to as the design matrix, where each row contains the data for one individual. $\mathbf{x_i}^T$ is thus the vector of features for individual i.

The scores for the entire dataset are given by :

$$\mathbf{s} = (s_1, ..., s_n)^T = X\beta.$$

4.2.3 Link function

The link function needs to be monotonic and differentiable. One possibility is the canonical link function. This is simply found by taking the inverse of the first derivative of $b(\theta)$. For the Bernoulli case, a suitable choice is the logit function :

$$g(\mu) = \log \frac{\mu}{1-\mu} = \theta = X\beta.$$

4.2.4 Optimization

To find the optimal parameters, the maximum likelihood approach can be used. In this approach, we take the logarithm of $f(y;\theta)$, which in matrix notation is :

$$l = \log(L) = y^T A^{-1}\theta - \{b(\theta)^{1/2}\}^T A^{-1} b(\theta)^{1/2} + K,$$

where θ and $b(\theta)$ are vectors, and K is a constant that does not interfere with optimization. To find the $\hat{\beta}$'s, we take the first derivative of l and replace θ with $X\beta$. The resulting formula then needs to be maximized :

$$X' A^{-1}(y - b'(X\beta)) = 0.$$

However, this is not linear, therefore a non-linear optimization technique such as Newton-Raphson's algorithm or the Iterative Weighted Least Squares method is required.

4.3 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding known as t-SNE is an extension of Stochastic Neighbor Embedding that was introduced by van der Maaten & Hinton (2008). It is a dimensionality reduction technique that tries to preserve meaningful local structure while also conserving the global structure of high-dimensional data, the mapping is then used to visualize the data into a 2D or 3D plot. As it preserves global structure, it is often used to detect potential clusters in the data even if it is not initially a clustering technique.

SNE, the algorithm t-SNE is initially inspired from, has a crowding problem where points that are close in a high-dimensional space are represented really close together in the

low-dimensional space and points that are far apart are not repulsed enough. Therefore, points form a crowd in the lower dimensional space. t-SNE solves this by introducing the Student t-distribution to represent pairwise distances in LD. This way, a distance in the low-dimensional space can never be smaller than the one in the higher dimensional space. More precisely, distances in the HD space are converted into Gaussian probabilities. In the LD space, we use a heavy-tailed distribution to represent those distances into probabilities : the Student t-distribution with one degree of freedom.

Also, SNE works with a cost function that uses the Kullback-Leibler divergence to compare the conditional probabilities $p_{j|i}$ and $q_{j|i}$. This represents the probability that a certain point x_i would pick x_j as its neighbor. t-SNE works with a symmetrized version of SNE in order to simplify the cost function. The goal is thus to minimize the KL divergence between the joint distribution of p_{ij} and q_{ij} :

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

A distance in the high-dimensional space is thus first converted to a conditional probability that x_i would pick x_j as its neighbor. Distances are represented as probabilities following a Gaussian distribution that is centered around x_i with σ_i the variance of the distribution around x_i. We get

$$p_{j|i} = \frac{\exp\big(-||x_i - x_j||^2/2\sigma_i^2\big)}{\sum_{k \neq l} \exp\big(-||x_k - x_l||^2/2\sigma_i^2\big)}$$

The joint distribution p_{ij} is then obtained by $\frac{p_{j|i}+p_{i|j}}{2n}$ in order to avoid having big distances that have such a small probability that they don't even contribute to the cost function.

For the distances in low dimension using the Student t-distribution, we get :

$$q_{ij} = \frac{\big(1 + ||y_i - y_j||^2\big)^{-1}}{\sum_{k \neq l} \big(1 + ||y_k - y_l||^2\big)^{-1}}$$

Minimization of the cost function happens using gradient descent. For improving the results, two tricks are added to the process. The first one is early compression : at the beginning of the optimization, data points are forced to stay close together in order to find clusters by adding a L_2 penalty term that is removed after a certain number of iterations of the algorithm.

Also at the beginning of the optimization, we introduce early exaggeration. It implies that all p_{ij}'s are multiplied for example by 4. All distances are thus greater and compensation is needed from the q_{ij}'s in order to match those bigger distances. This causes clusters to spread far from other clusters leaving blank space in the LD map and allowing clusters to position themselves meaningfully compared to the other clusters.

5 Case Study

This section will present the actual implementation of the case study. We will apply various supervised and unsupervised techniques to the dataset introduced earlier in this work. This will involve working with either the original dataset, a generated set or a reduced set that has been encoded using the variational autoencoder's encoder model.

As explained in the introduction, we will perform clustering on the mapped data and the generated data, and regression on this first. To evaluate the effectiveness of these techniques, the clustering results will be assessed against those from clustering using the Burt matrix derived from the original data. Meanwhile, the regression results will be compared against those from a regression performed on the original dataset after one-hot encoding. Hyperparameters for all the models will be chosen once the comparison metric and the training technique are defined.

5.1 K-fold Cross Validation

To train a model, it is necessary to work with a training set, which is used to calibrate the model initially, and then assess the performance of the model using a new set called a test set. This is a set of data that was never seen by the model during training, which allows for measuring the performance of the model on unseen data. Indeed, there is a high risk of overfitting if we use one single set of data for both training and testing. Overfitting occurs when the model is too well trained on training data, capturing noise as if it was part of the meaningful pattern. As a result, although the model seems to work really well on the training data, it fails to generalize to new unseen observations, leading to large errors. The rule of thumb is to work with 80% of the initial set for training and test the model with the remaining 20%.

Running a model with the same set of parameters multiple times might yield results that are sometimes relatively far apart since each run of the model will predict different probabilities. Therefore, we work with k-fold cross-validation, where the initial dataset is divided into k subsets that are subsequently split into a training and a test set (Denuit et al., 2019b). By doing this, we get to train the model k times and take the average of all the errors. This technique is time-consuming and computationally expensive, which is why we choose k to be equal to 5. The error resulting from the k-fold cross-validation is given by

$$CE_K = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in \mathcal{B}_k} E\left(y_i, \hat{y_i}^{(k)}\right)$$

where E is the chosen error function. Explanations about this follow on the next page.

5.2 Assessment of the models

A comparison and goodness of fit criterion are needed to choose the optimal hyperparameters for each model and then compare the different approaches. Many options exist for this, such as the mean squared error, the accuracy or the precision.

However, these are sometimes suboptimal depending on the context. This is the case here, as our task is to classify observations into two classes by predicting the probability that a certain passenger is satisfied or not. The closer the predicted probability is to 0, the more probable it is that the traveler was not satisfied. On the other hand, if the prediction is close to 1, the probability is high that the traveler was satisfied. A better criterion for such a problem is cross-entropy (Hainaut, 2023).

The equation used to compute the average binary cross-entropy is the following :

$$\text{Mean Binary Cross-Entropy} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

where y_i is the true label, N the number of observations in the dataset, and $\hat{y}_i$ represents the probability that observation i belongs to class 1 (satisfied). When the true label is 0, only the second term of the formula is taken into account. Conversely, when the true label is equal to 1, only the first term remains. The first term is thus the loss when the traveler is satisfied and the second term is the loss when the traveler is dissatisfied. This is to be computed over all observations of the dataset, then the sum of all losses is divided by the number of elements to obtain the average.

The obtained loss represents the dissimilarity between the true labels and the predicted probabilities. The lower the cross-entropy, the better the model is at predicting probabilities that are close to the actual labels.

Note that the cross-entropy explained here is not the same as the one used as reconstruction loss in the autoencoder. The formula is the same, but they are not used in an identical context and are therefore not comparable. The cross-entropy used in the neural network returns the dissimilarity between the input and the output of the network. It is thus comparing X and its reconstruction, while the cross-entropy as a goodness of fit measure is used to compare the true labels Y and the predictions $\hat{Y}$.

5.3 Architecture of the variational autoencoder

For both the clustering and the regression task, a variational autoencoder is built and trained. The encoder network is then used to reduce the dimensionality of the data, which serves as the input for the K-means and the generalized linear model. For both approaches, the architecture of the variational autoencoder is the same, meaning that the number of hidden layers and the activation functions in each layer are identical. However, hyperparameters such as the number of neurons or the batch size may vary. To create the most optimal model for each prediction technique, each model was trained with different sets of parameters. For each technique, the set yielding the smallest mean cross-entropy will be kept.

It is important to note that training the model was computationally expensive. Therefore, some hyperparameters were tuned "by hand" by experimenting with variations and choosing the optimal one, while others were tuned through a grid search. More precisely, the parameters that were fine-tuned through the grid search are : the batch size (`batchsize`), the learning rate (`lr`) for the optimizer, the number of neurons in the latent layer (`latent_dim`) and the number of neurons in the first layer of the encoder (`d1_encoder`) and the last layer of the decoder (`d2_decoder`) (`intermediate_dim`).

The other hyperparameters were either found by hand before the grid search or chosen after fine-tuning the four parameters mentioned above. For example, the number of clusters will be decided later since a larger number of clusters is assumed to yield better results. Indeed, with more clusters, it would be easier to have clusters that are more representative of each data point. We thus expect that, when put into a grid search, the model with the highest number of clusters would outperform the others. A plot showing the mean cross-entropy for each possible number of clusters will help us decide on the number of clusters.

For the number of epochs, we will look at a graph plotting the average cross-entropy in function of the epochs and the total loss of the model across the epochs, which will help us to decide when to stop the training. For the grid search, the number of clusters will be set to 10 and the number of epochs to 100.

Among the choices made before the exhaustive training, we have the optimization technique, the loss function, the number of layers for both the encoder and the decoder, the activation function for each layer, and the number of neurons on the other intermediate layers (`d2_encoder`, `d3_encoder` and `d1_decoder` as they were defined later on). The choice was made both by consulting the literature and trying several combinations.

First of all, we will define the loss function that will be used to train the VAE. The loss function of the variational autoencoder, as explained in the theoretical part of this work, is composed of the Kullback-Leibler divergence and a reconstruction loss function. This first term is defined with the following formula (Hainaut, 2023) :

$$D_{KL}(q_\phi(z|x_i)\|p_\theta(z)) = \frac{1}{2}\sum_{i=1}^{p}\left(\mu_i^2(x) + \sigma_i^2(x) - (1 + \log(\sigma_i^2(x)))\right)$$

It represents the encoding error. The second term constitutes the reconstruction error. The function we will use for that part is the binary cross-entropy, which is the same function that we introduced as the assessment approach, but used in another context. The difference is that, now, the sum of the losses is taken instead of the average. The formula is :

$$\text{Binary Cross-Entropy} = -\sum_{i=1}^{N}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right].$$

For the optimizer, we experimented with options such as RMSprop and ADAM. RMSprop was giving better results on our data, thus we opted for that one.

Additionally, for the number of layers, we added extra layers until adding a new one no longer improved the results. Therefore, we ended up working with 3 hidden layers in the encoder and 2 in the decoder.

Also, for the activation function, since our goal is to reconstruct the input, which is binary (composed only of 0s and 1s), a good practice is to use the sigmoid activation function on the last layer. The values that are returned by this function are between 0 and 1, which potentially reduces the loss. For the other layers, we used the ReLU activation function as it produced the best results.

Finally, for the intermediate layers (`d2_encoder`, `d3_encoder` and `d1_decoder`), we chose to work with 20 neurons on `d2_encoder` and 15 neurons on the other two layers. This was again chosen through trial and error and appeared to give satisfactory results.

Here follows the architecture explained layer by layer, gathering all the choices that were explained just above :

1. The **Encoder network** is composed of an input layer and 3 hidden layers. It is the part of the variational autoencoder that compresses the input data into a lower-dimensional representation. Here follows an explanation on each element :

- Input layer which defines the input shape for the VAE with as shape (113,), 113 being the number of input variables.
- `d1_encoder`: Dense layer with `intermediate_dim` neurons and a ReLU activation function. `intermediate_dim` is a hyperparameter of the model. In the training, it was assigned the values 20, 30 and 40.
- `d2_encoder`: Dense layer with 20 neurons and a ReLU activation function. The neurons in this layer and in `d3_encoder` are fixed, they are not hyperparameters, and have been chosen through trials.
- `d3_encoder`: Dense layer with 15 neurons and a ReLU activation function.
- `z_mean`: Dense layer producing the mean of z, the latent space representation. The shape of this layer is (`latent_dim`,), where `latent_dim` is also a hyperparameter with a domain of [5, 10, 15].
- `z_log_var`: Dense layer producing the log variance of z. The shape of this layer is the same as the one for `z_mean`.

2. The **Sampler Layer** is the layer that samples ϵ from `z_mean` and `z_log_var` to create the latent space representations. This is the application of the reparametrization trick. The outputs of the encoder network are means and log variances which cannot directly be used for other tasks. It is the role of the Sampler Layer to compute the actual values of the latent representations that can further be used as embeddings for clustering, regression, etc.

3. The **Decoder network** has as input the latent representations and is constituted of 2 hidden layers. The outputs of this network are the reconstructions of the inputs of the VAE.

   - `d1_decoder`: Dense layer with 15 neurons and ReLU activation function.
   - `d2_decoder`: Dense layer with `intermediate_dim` neurons and ReLU activation function. `intermediate_dim` is the same parameter as for `d1_encoder`.
   - `outputs`: Dense layer producing the output data, with the same dimension as the input data (113), and using the sigmoid activation function in order to have outputs located between 0 and 1.

The encoder and the decoder can be used independently for reducing the dimension (encoder) or generating data (decoder), or together to reconstruct a certain input.

The last task before letting the grid search operate is to decide the domain for each hyperparameter. The domain consists of the values that will be tested for each hyperparameter.

The batch size will define the size of the dataset utilized in one iteration of the stochastic gradient descent optimization algorithm. While a larger batch size will accelerate the algorithm, training a model with a smaller batch size can help with model generalization as it introduces more noise into the model updates (Jamotton & Hainaut, 2023). Therefore, we will try different values : 200, 500, 1000 and 10000. This will allow us to see if a smaller batch size offers advantages over a larger one in this context.

For the learning rate, our initial experimentation revealed that learning rates above 0.001 don't yield satisfactory results. The domain is thus limited to 0.0001 and 0.001 (the default rate).

Regarding the number of neurons in the different layers, the Latent Layer acts as the bottleneck. It is thus necessary to have a low number of neurons to induce dimensionality reduction. While clustering seemed to perform better with fewer dimensions, regression required a certain amount of information to construct a meaningful model. Thus, we consider 5, 10 and 15 neurons for the Latent Layer.

Moreover, the choice for the Intermediate Layers is 20, 30 and 40. This choice is motivated by suboptimal results when working with a higher number of neurons for the first hidden layer, and the objective is to allow a sufficiently high reduction and learning across layers.

Table 6 summarizes the domain for each hyperparameter.

Hyperparameters	Domain
Batch size	[200; 500; 1000; 10000]
Learning rate	[0.0001; 0.001]
Number of neurons on Intermediate Layers (`d1` of encoder and `d2` of decoder)	[20; 30; 40]
Number of neurons on Latent Layer	[5; 10; 15]

Table 6: Hyperparameters' domain for the VAE

5.4 Clustering

5.4.1 VAE-encoded dataset

Before starting the training, we will briefly explain the steps followed. First, we train a variational autoencoder. Then, the training and test sets are compressed using the encoder network, which returns arrays of means and variances. Using the Sampler Layer, we sample ϵ and transform it into z_{train} and z_{test}. These latent sets are then used for training the K-means algorithm. The algorithm is first trained with the compressed training set, and labels are predicted for the compressed test set. In this case, the cross-entropy is the difference between the true satisfaction and the satisfaction of the cluster to which an individual is assigned. For example, if a cluster has a probability for being satisfied equal to 0.14 and customer i is assigned to that cluster, we compute the dissimilarity between 0.14 and the true satisfaction of customer i. The mean cross-entropy loss is then calculated over all individual losses.

Now let's return to the fine-tuning. The grid search is initialized with the average cross-entropy of the regression as the metric, and we can start running all combinations with a 5-fold cross-validation. The number of epochs for the autoencoder is 100, and the clustering is initialized with 10 clusters.

Also, for the K-means algorithm, we used a random assignment of the initial cluster centers, which is repeated 20 times. Thus the algorithm will run 20 times with different initializations of clusters and keep the best one. Additionally, we chose to stop the algorithm after a maximum of 300 iterations, which is the default setting. The five best models using the grid search are displayed in Table 7.

	Batch size	Intermediate dimension	Latent dimension	Learning rate	Cross-entropy
1	1000	40	5	0.001	0.42278
2	1000	40	15	0.001	0.43417
3	1000	20	10	0.001	0.43685
4	200	40	5	0.001	0.43941
5	500	40	5	0.001	0.44042

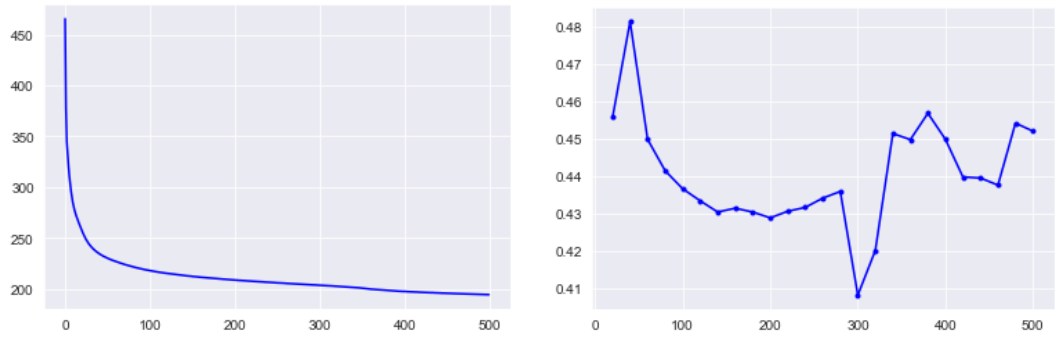
Table 7: Results top 5 for K-means

From Table 7, we note that a learning rate of 0.0001 as well as an intermediate dimension of 30 don't seem to be efficient for this task. The optimal model has a batch size of

1000, similar to the regression, a learning rate of 0.001 (the default rate), an intermediate dimension of 40, and a latent dimension of 5, as we expected before training.

Now the last parameters we have to fine-tune are the number of epochs and number of clusters. Taking a too low number of epochs might result in undertraining and thus having a model that does not capture all the relevant information. Conversely, training the model for too long makes it learn noise as part of the meaningful pattern, which doesn't generalize on new data.

Figure 7a shows the total loss over 500 epochs and Figure 7b the mean cross-entropy as a function of the number of epochs used for training the model. In Figure 7a, we see that around the 30th epoch, the loss starts to stabilize indicating that the model converges from that point . Furthermore, Figure 7b shows a surprisingly clear drop in the average cross-entropy around the 300th epoch. The model will thus be trained for 300 epochs.



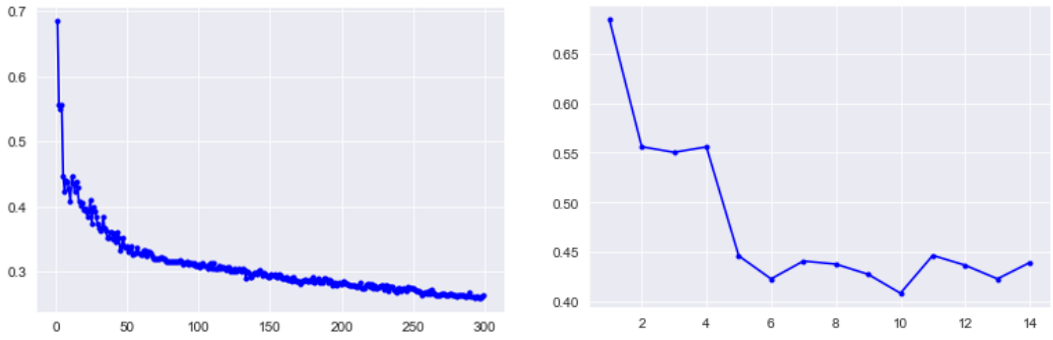
(a) (b)

Figure 7: (a) The total loss when training a variational autoencoder with the set of hyperparameters for the K-means across epochs. (b) The mean cross-entropy across epochs.

Regarding the number of clusters, we take a look at Figure 8. The left plot shows the mean cross-entropy over the number of clusters going from 1 to 300, and the right plot is a zoomed in version showing clusters from 1 to 14.

It is interesting to note that, as we predicted in a preceding section, the larger the number of clusters, the lower the mean cross-entropy. However, in many cases, we prefer a compact representation with fewer clusters. In our case, we will consider a maximum of 14 clusters and from the graph on Figure 8b, 10 clusters seems to be the optimal quantity with a mean cross-entropy of 0.4081.

Variational autoencoders aim at reducing the dimensionality of the data and proposing



(a) (b)

Figure 8: (a) The mean cross-entropy from 1 to 300 clusters (b) and 1 to 14 clusters.

disentangled latent variables. Before presenting the clusters, we will take a look at the correlation matrix of the latent variables z. This is available in the appendix (see Figure 18). Most of the latent variables are relatively uncorrelated except for X2 that has correlations between 0.29 and 0.44 with all other latent variables.

To show the representative point of each cluster, we will compute the mode value for each variable, which will give us representative profiles of the dataset. The following tables (8, 9 and 10) display the 10 profiles in order of satisfaction, going from the least to the most satisfied. Note that the names of the variables have been abbreviated to fit in the tables.

	Gender	Customer Type	Travel Type	Class	Inf. wifi	Time Conv.	Online booking	Gate loc.	Satisfaction
1	Male	Loyal	Personal	Eco	2	2	2	3	0.0217
2	Female	Loyal	Personal	Eco	2	4	2	3	0.0561
3	Male	Loyal	Business	Eco	3	4	4	3	0.2681
4	Male	Loyal	Personal	Eco	1	4	1	3	0.3109
5	Female	Loyal	Business	Business	3	4	3	3	0.3555
6	Female	Loyal	Business	Business	2	4	3	3	0.4040
7	Male	Loyal	Business	Eco	5	4	3	3	0.5417
8	Male	Loyal	Business	Business	4	1	1	1	0.9168
9	Female	Loyal	Business	Business	5	4	4	3	0.9772
10	Female	Loyal	Business	Business	2	2	2	2	0.9792

Table 8: Main profiles, VAE encoding (part 1/3)

	F&D	Online boarding	Seat Comfort	Entertain-ment	Onboard service	Leg room	Baggage handling	Satisfa-ction
1	2	2	1	2	3	3	4	0.0217
2	4	2	4	4	4	3	4	0.0561
3	3	3	3	3	3	3	4	0.2681
4	2	1	2	2	4	4	4	0.3109
5	2	4	4	3	3	3	3	0.3555
6	4	4	4	2	2	2	2	0.4040
7	5	5	5	5	2	2	4	0.5417
8	4	4	4	4	4	4	4	0.9168
9	4	4	4	5	5	5	5	0.9772
10	4	4	4	4	4	4	4	0.9792

Table 9: Main profiles, VAE encoding (part 2/3)

	Check-in service	Inflight service	Cleanliness	Age	Flight distance	Departure delay	Arrival delay	Satisfa-ction
1	3	4	2	19-30	0-1000	0-5	0-5	0.0217
2	4	4	4	19-30	0-1000	0-5	0-5	0.0561
3	3	4	3	19-30	0-1000	0-5	0-5	0.2681
4	4	4	2	19-30	0-1000	0-5	0-5	0.3109
5	4	3	4	51-64	0-1000	0-5	0-5	0.3555
6	3	2	4	41-50	0-1000	0-5	0-5	0.4040
7	3	4	5	19-30	0-1000	0-5	0-5	0.5417
8	4	4	4	41-50	0-1000	0-5	0-5	0.9168
9	3	5	3	41-50	0-1000	0-5	0-5	0.9772
10	3	4	4	41-50	0-1000	0-5	0-5	0.9792

Table 10: Main profiles, VAE encoding (part 3/3)

The first noteworthy point is that customers traveling for personal reasons appear to be less satisfied, which aligns with what we observed from the barplots in section 2.1.1. Indeed, three out of the four least satisfied customers are traveling for personal reasons. Additionally, the clusters verify the hypothesis that people traveling in Eco class are less satisfied. No representative of the Eco Plus class is present, but this was expected as this class is underrepresented in the original dataset.

Similarly, for the variables *Customer Type*, *Flight distance*, *Departure delay* and *Arrival delay*, only one modality of each variable is represented. This is because of the imbalanced distribution of the modalities in the dataset. Thus, the clustering doesn't help us to validate or refute the hypothesis that a long flight results in more satisfied customers.

In terms of age, we noted that older passengers were more satisfied, which also appears from the clustering. The four least satisfied profiles fall within the 19 to 30 age bracket, while the three most satisfied are aged between 41 to 50 years old.

As for the service grades, we note that the ratings for the *Baggage handling* and the *Inflight service* are higher than for the other variables. Conversely, *Inflight Wifi service*, *Ease of Online Booking* and *Gate location* show lower ratings, which is consistent with our previous findings from the barplots.

Overall, the identified clusters mirror the insights previously outlined. We only note an underrepresentation of variables with imbalanced distributions.

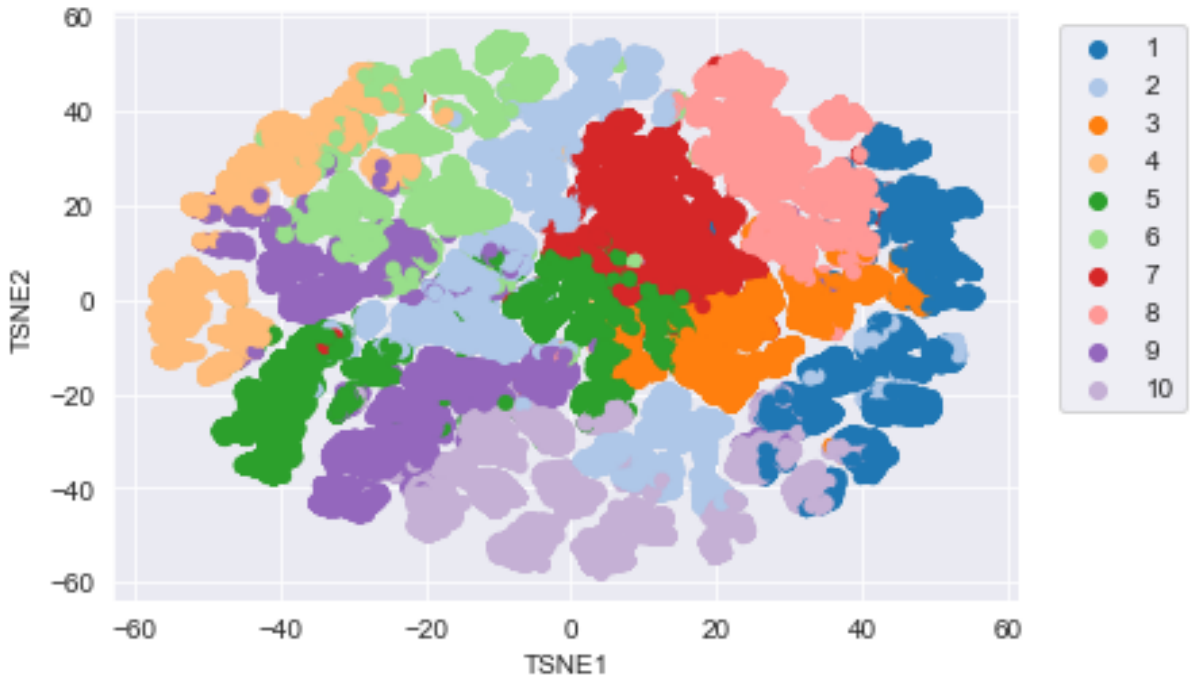


Figure 9: 2D t-SNE plot colored by clusters resulting from the K-means.

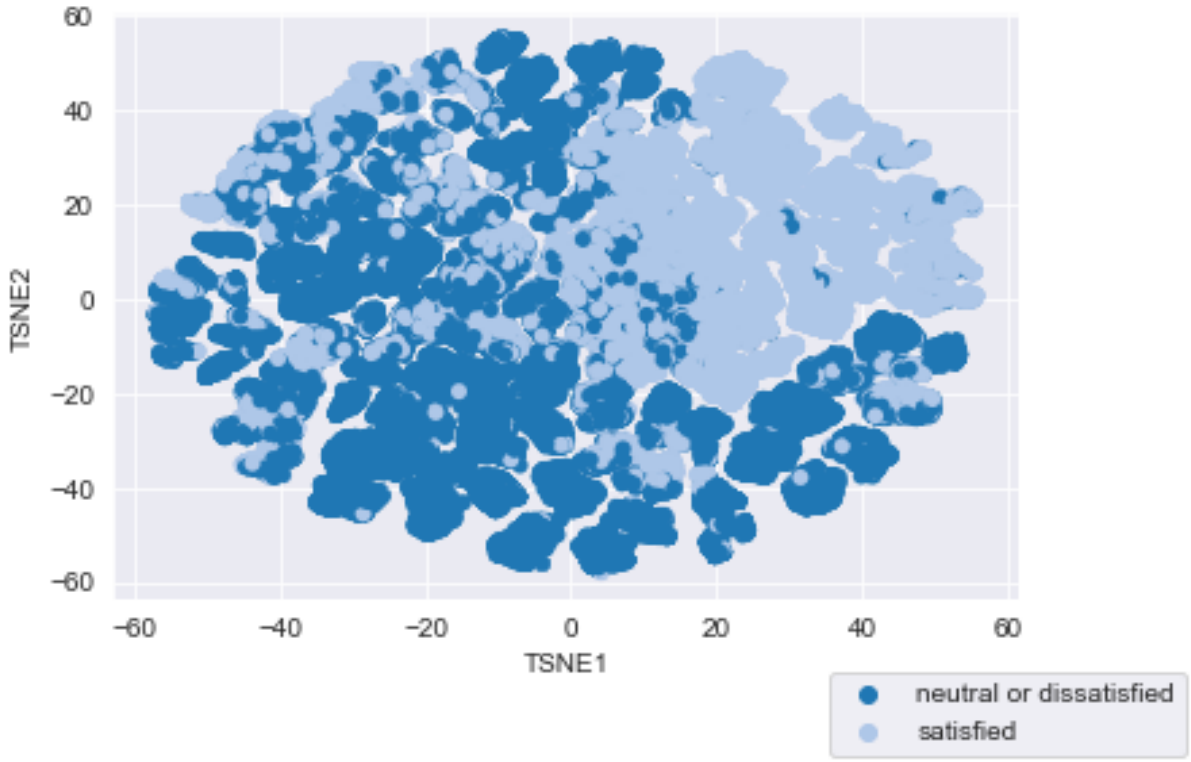


Figure 10: 2D t-SNE plot colored by *satisfaction*.

The t-SNE algorithm helps us to visualize high-dimensional data in lower dimensions. We have represented the embedding variables into 2D and 3D space to find out if the latent space reveals some interesting structures such as clusters.

Data points on Figure 9 are colored by cluster. We notice that although there is no distance between the clusters, data points are well grouped with other points of their cluster except for the clusters 2, 5 and 9 that are split in two.

Additionally, when comparing Figure 9 and Figure 10, we observe that spots where satisfied and neutral/dissatisfied customers are mixed mostly coincide with clusters 4, 5 and 6 that have a probability of satisfaction between 0.3 and 0.4. These are clusters that represent passengers for whom it is more difficult to detect if they will be satisfied or not.

5.4.2 Burt encoded dataset

The second technique we implement for clustering is the Burt matrix. As explained in section 2.2.3, it is an approach to quantify distances between categorical variables.

Similarly to the K-means using the VAE, we will encode the dataset using the Burt matrix

and divide the set into a training (80%) and a test set (20%). The training set will serve for training the K-means algorithm and customers from the test set who were not used for the training will be assigned to a cluster. After that, the mean cross-entropy of the set is calculated. Note that the parameters for the clustering algorithm are the same as with the variational autoencoder : 20 random initializations of cluster centers with a maximum of 300 iterations in order to find the optimal clustering.

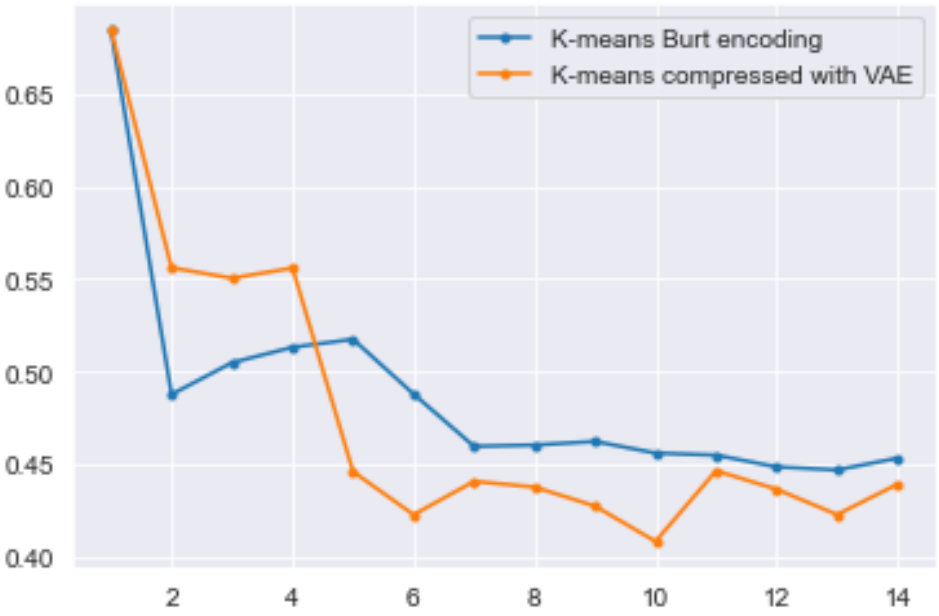


Figure 11: Comparison of the cross-entropy with Burt encoding and VAE

Figure 11 illustrates the evolution of the mean cross-entropy with the number of clusters increasing from 1 to 14 for both VAE encoded data and Burt encoded data. Although the Burt encoding outperforms the VAE encoding when the number of clusters is located between 2 and 4, we see that the latter takes the lead starting from 5 clusters. The best result achieved with the Burt matrix is a mean cross-entropy of 0.4468 with 13 clusters compared to 0.4081 for the variational autoencoder with 10 clusters. This underscores the superiority of the deep neural network approach. Nonetheless, it is worth mentioning that Burt encoding yields interesting results while being easy to use and implement, and computationally inexpensive.

Again, we will have a quick look at the main profiles (see Tables 11, 12 and 13) of customers detected using the clustering. We have decided to work with 10 clusters to make the comparison easier. The cross-entropy using the Burt matrix equals 0.4559, which is almost 0.05 higher than with the VAE.

	Gender	Customer Type	Travel Type	Class	Wifi	Time Conv.	Online booking	Gate	Satisfaction
1	Male	Loyal	Business	Eco	1	4	2	3	0.1418
2	Male	Loyal	Business	Eco	2	4	2	3	0.1586
3	Female	Loyal	Business	Eco	3	4	3	3	0.2001
4	Female	Loyal	Business	Eco	1	1	1	3	0.2250
5	Female	Loyal	Business	Business	2	4	2	3	0.2780
6	Male	Loyal	Business	Eco	4	4	3	3	0.3204
7	Male	Loyal	Business	Eco	3	4	3	3	0.3470
8	Female	Loyal	Business	Business	4	4	4	4	0.8893
9	Female	Loyal	Business	Business	5	5	5	5	0.9383
10	Female	Loyal	Business	Business	5	5	1	1	0.9602

Table 11: Main profiles, Burt encoding (part 1/3)

	F&D	Online boarding	Seat Comfort	Entertainment	Onboard service	Leg room	Baggage handling	Satisfaction
1	1	1	1	1	4	3	4	0.1418
2	2	2	2	2	3	2	4	0.1586
3	3	3	3	3	3	3	3	0.2001
4	3	3	3	1	1	1	1	0.2250
5	4	4	4	2	2	2	2	0.2780
6	4	4	4	4	4	4	4	0.3204
7	5	5	5	5	3	3	4	0.3470
8	4	4	4	4	4	4	4	0.8893
9	4	4	4	4	4	4	4	0.9383
10	5	5	5	5	5	5	5	0.9602

Table 12: Main profiles, Burt encoding (part 2/3)

	Check-in service	Inflight service	Cleanliness	Age	Flight distance	Departure delay	Arrival delay	Satisfaction
1	4	4	1	19-30	0-1000	0-5	0-5	0.1418
2	4	4	2	19-30	0-1000	0-5	0-5	0.1586
3	3	3	3	19-30	0-1000	0-5	0-5	0.2001
4	3	1	3	51-64	0-1000	6-60	6-60	0.2250
5	3	2	4	41-50	0-1000	0-5	0-5	0.2780
6	4	4	4	19-30	0-1000	0-5	0-5	0.3204
7	4	4	5	19-30	0-1000	0-5	0-5	0.3470
8	4	4	4	41-50	0-1000	0-5	0-5	0.8893
9	4	4	4	41-50	0-1000	0-5	0-5	0.9383
10	5	5	5	41-50	0-1000	0-5	0-5	0.9602

Table 13: Main profiles, Burt encoding (part 3/3)

The first notable observation is the unique outcome for the variables *Customer Type* and *Flight distance*, which is similar to the outcome of the previous technique. However, *Travel Type* also exhibits only one modality, leading to a reduction in insights on the impact of the travel type on satisfaction when utilizing the Burt matrix. Conversely, *Departure delay* and *Arrival delay* now illustrate the influence of delays on satisfaction. Cluster 4 specifically showcases a delay ranging from 6 to 60 minutes, coupled with an older passenger (51-64). As we had previously highlighted, the age influences satisfaction positively, however the delay results in a lower satisfaction for that cluster illustrating the negative impact a delay can have on satisfaction.

Here again, we see the impact of the age on the probability of being satisfied, the majority of unsatisfied passengers fall within the 19 to 30-year-old range while passengers aged 41 to 50 present the highest probability of satisfaction. Additionally, the Eco class again demonstrates a lower satisfaction than the Business class.

Regarding the service variables, conclusions are harder to draw. Table 14 summarizes the average ratings for all service variables from the original dataset as well as from the clustering using the VAE and the Burt matrix. Differences between the original data and the clusters' means are also displayed to depict the deviance that occurs from the clustering.

Departure/Arrival time convenient (3.9) and *Checkin service* (3.8) have the highest average ratings with Burt, which doesn't align with our hypotheses and the observations made with the VAE. Variables with the lowest ratings are *Inflight wifi service*, *Ease of*

Online booking and *Gate location* with means around 3 when working with Burt, which corresponds to the hypotheses (between 2.73 and 2.98). Additionally, *Inflight entertainment* and *Leg room service*, which were rather on the higher end of the scale with means equal to 3.35 and 3.36 in the original dataset, also show lower ratings.

We thus note that the ratings produced by the clustering combined with the Burt matrix are relatively far from the reality of the original dataset. Some large differences are noticeable, with a maximum of +0.84, while for the variational autoencoder, the means don't differ further than —0.28— from the original dataset means. The mean difference when using the VAE is 0.16 while this goes up to 0.26 for Burt. The variational autoencoder thus more accurately reflects the ratings given by the customers.

Variable	Mean Original	Mean VAE	Mean Burt
Inflight wifi service	2.73	2.9 (+0.17)	3 (+0.27)
Ease of Online booking	2.76	2.5 (-0.26)	3 (+0.24)
Gate location	2.98	2.7 (-0.28)	3.1 (+0.12)
Departure/Arrival time convenient	3.06	3.3 (+0.24)	3.9 (+0.84)
Food and drink	3.2	3.4 (+0.2)	3.5 (+0.3)
Online boarding	3.25	3.3 (+0.05)	3.5 (+0.25)
Cleanliness	3.29	3.5 (+0.21)	3.5 (+0.21)
Checkin service	3.31	3.4 (+0.19)	3.8 (+0.49)
Leg room service	3.35	3.3 (-0.05)	3.1 (-0.25)
Inflight entertainment	3.36	3.4 (+0.04)	3.1 (-0.26)
On-board service	3.38	3.6 (+0.22)	3.3 (-0.08)
Seat comfort	3.44	3.5 (+0.06)	3.5 (+0.06)
Baggage handling	3.63	3.8 (+0.17)	3.5 (-0.13)
Inflight service	3.64	3.8 (+0.16)	3.5 (-0.14)

Table 14: Mean values of service variables of original data, clustering with VAE and clustering with Burt

5.5 Regression

5.5.1 VAE-encoded dataset

As we did for the clustering, we will train a variational autoencoder. We first train the generalized linear model with the latent training set and further predict the probability that each passenger of the mapped test set is satisfied. The mean cross-entropy is then calculated for these predicted probabilities, comparing them with the actual satisfaction of the individuals. Since mean cross-entropy is the measure we want to minimize, it will serve to choose the optimal set of hyperparameters. We recall that we work with a k-fold cross-validation, so this entire process will be repeated five times.

Again, each of the models was trained for 100 epochs. Table 15 ranks the five best performing models.

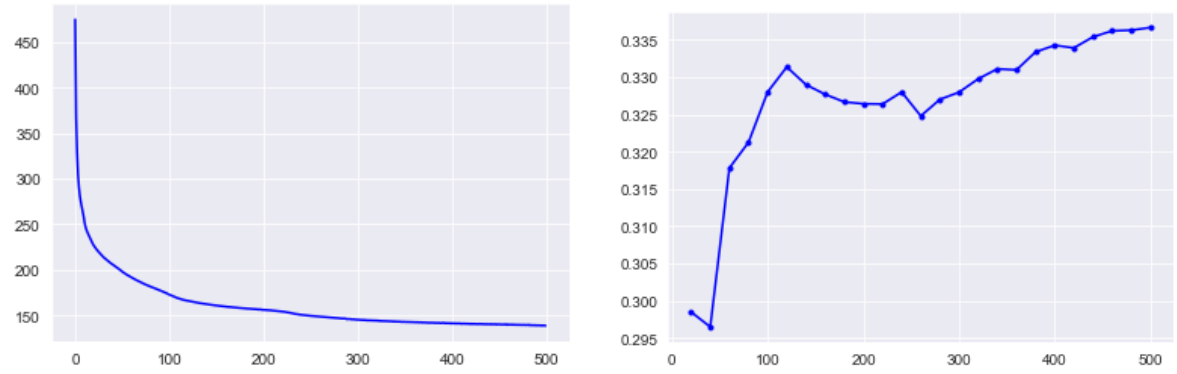
	Batch size	Intermediate dimension	Latent dimension	Learning rate	Cross-entropy
1	1000	30	15	0.001	0.3084
2	500	40	15	0.0001	0.311
3	200	40	15	0.0001	0.31226
4	1000	40	15	0.0001	0.31981
5	500	40	10	0.0001	0.32184

Table 15: Results top 5 for the GLM

From this table, we see that the best model is one with a batch size of 1000, 30 neurons on the Layers `d1_encoder` and `d2_decoder`, 15 neurons on the Latent Layer, and a learning rate of 0.001. We see that 20 for `intermediate_dim` was not a good option, it does not come up in the best performance. Also, a batch size of 10,000 and a Latent Layer with 5 neurons look suboptimal for this task.

Now the last hyperparameter that we need to fine-tune is the number of epochs. We thus retrain the model on a higher number of epochs : 500.

We see in Figure 12a that the total loss drops suddenly around the 10th epoch and starts to stabilize from the 100th epoch onwards. Also, in Figure 12b, we have plotted the mean cross-entropy resulting from the regression in function of the number of epochs. There is a drop around the 40th epoch. Therefore, the model will be trained for 40 epochs, which yields an average cross-entropy of 0.2965. This is interesting since, for the



(a) (b)

Figure 12: (a) The total loss when training a variational autoencoder with the set of hyperparameters for the GLM across epochs. (b) The mean cross-entropy across epochs.

clustering, it was necessary to train for 300 epochs. We also note that the clustering would be outperforming the regression with a high number of clusters. Indeed, we recall from Figure 8a that the mean cross-entropy was decreasing beyond 0.29 when the number of clusters was higher than 200.

As we did in the previous section, we analyze the independence of the latent variables proposed by this model (see Figure 19). Unfortunately, the results are less satisfying with a large proportion of correlations above 0.1. This can therefore probably explain the lack of performance in the regression, leading to a mean cross-entropy of 0.29.

The results of the GLM can be found in the appendix (see Appendix 6.3). We note that all the variables are significant with a p-value smaller than 0.01. Also, all coefficients are within a relatively small range, with the highest one being 1.0620 for x_4 and the lowest one -0.8296 for x_7. Furthermore, all coefficients bear almost the same amount of information, meaning that they are on an equivalent scale (approximately between |0.1| and |1|).

5.5.2 One-hot encoded dataset

A useful benchmark is to compare the results of the just-presented technique with those of a regression using the original dataset with one-hot encoding.

Some extra preprocessing was required for training the GLM with the unmapped dataset. Initially, when one-hot encoding a dataset for regression, it is necessary to drop one level of each categorical variable. However, some of the dropped levels had only few individuals causing the model to fail to converge. Thus, besides dummyfying the dataset, it was necessary to indentify which variables posed this problem. After investigation, the variables *Inflight entertainment*, *On-board service*, *Gate location*, *Inflight wifi service*,

Checkin service, *Inflight service*, *Seat comfort*, and *Cleanliness* were found problematic.

After removing these variables, another issue arose. The coefficients for the variable *Inflight wifi service* were considerably high and inconsistent compared to the rest, although the mean cross-entropy for this model was good. Even though this variable didn't have issues with a low number of individuals in the dropped level, we decided to remove an extra level. This led to a slight performance loss, with the averaged cross-entropy increasing from 0.1778 to 0.2014. However, the results are more coherent and interpretable.

The results of the model can be found in Appendix 6.4. The average satisfaction is equal to 1.2, reflecting the satisfaction of a loyal female passenger traveling in business class for professional purposes, aged between 7 and 18. All service variables have received either a rating of 0 or 1, with the flight experiencing a delay of at most 5 minutes at departure and arrival, covering distances less than 1000 km.

Regarding the coefficients, *Gender* is not significant, nor are the coefficients of *Food and drink*, *Departure Delay in Minutes*, and some levels of other variables such as *Baggage handling* and *Flight distance*. They all present a p-value higher than 0.01. Nevertheless, we note that flights with distances between 1001-2000 and 4001-5000 have significant positive coefficients, meaning that a longer distance results in more satisfaction.

Additionally, *satisfaction* significantly decreases when the customer is disloyal (-3.02) or when the travel is for personal reasons (-3.96), and to a lesser extent, when flying in Eco (-0.63) or Eco Plus (-0.79) class. Moreover, longer delays at arrival result in lower satisfaction; for a 6-minute delay, *satisfaction* decreases by 0.71.

There is a trend indicating higher satisfaction for the variables like *Inflight entertainment*, *Inflight service*, *On-board service*, etc., with higher ratings. Furthermore, an *Inflight wifi service* that is given a rating of 5 results in an increase of 5.79 in satisfaction. Conversely, *Ease of Online booking* negatively influences *satisfaction*, suggesting that passengers are more satisfied when booking on-site.

Certain results are concerning. For example, *Gate location* and *Departure/Arrival time convenient* exhibit a lower satisfaction when the rating increases.

Finally, satisfaction across age categories is not linear. There is a peak of satisfaction for passengers aged 41- 50 (+0.36), followed by 19-30 (0.26) and 51-64 (0.19), while passengers aged 65+ show a decrease of satisfaction (-0.47).

5.6 Generation of data

We have now explored the dimensionality reduction abilities of the variational autoencoder. The model is also known for its generative capabilities, which we will now briefly discuss.

The generative model of the VAE is used to generate synthetic data. For this, a latent dataset needs to be sampled from a prior distribution. In our case, we will work with a sample drawn from a normal distribution : $\mathcal{N}(0, I)$. The latent $\mathbf{z}$ then serves as input for the generative model, the decoder neural network, which generates a sample of synthetic passengers.

The methodology for this part is as follows : first, we train a VAE with suitable hyperparameters to provide us high-quality great generated data, thus data that exhibit similar properties to the original dataset. The generalized linear model presented earlier will then be utilized to predict the satisfaction for each synthetic passenger. With this, we create a full synthetic dataset of customers with all their characteristics (age, gender, etc.), the ratings they supposedly gave, and their predicted satisfaction.

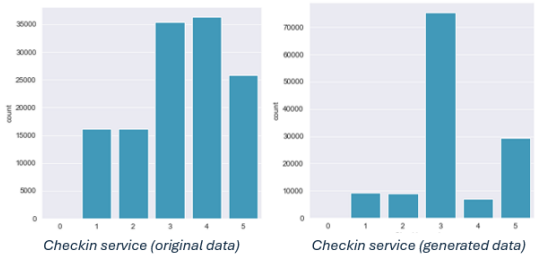
Next, this set is compressed using a second VAE, which is more suited for dimensionality reduction, and clustering is performed on the encoded data. That way, we obtain ten main profiles of synthetic passengers that can be compared with the real ones.

As mentioned, two different networks will be implemented and trained to obtain the best results. This is because the conventional variational autoencoder is not able to simultaneously generate high-quality data and reduce the dimensionality of the dataset by disentangling the latent space, as we have highlighted in section 3.3.

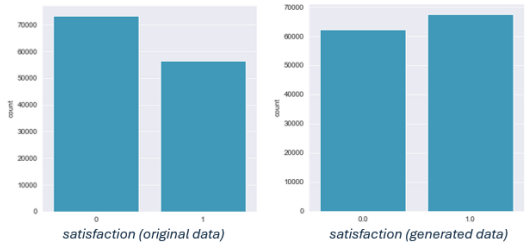
Note that the models' architecture was not modified from the one proposed earlier, but the number of epochs and the batch size were adapted. A smaller batch size allows for more noise and thus better data generation (Jamotton & Hainaut, 2023). Therefore, after some trial and error, we chose to work with a batch size of 30 and 40 epochs for the first model used for generating the new dataset. As for the encoding task, we opted for batches of 5000 items trained over 50 epochs.

The results are satisfactory but lack precision. We observe from Figure 20 and Figure 21 (in Appendix 6.5) that the modalities of the variables in the generated data seem to be approximately represented in the same quantities as the original data. However, some exception exist, such as the variable *Checkin service*. We see in Figure 13a that the VAE mostly generated instances with a rating of 3, while the distribution in the original dataset was more balanced. Rating 4 is surprisingly underrepresented, even though it is the most

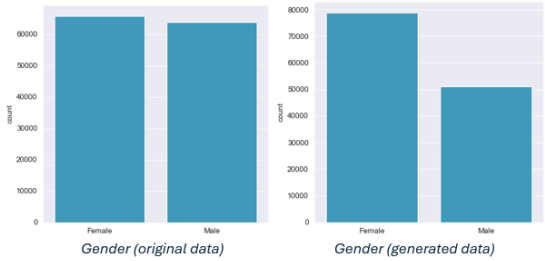
frequent rating in the true dataset for that variable.



(a) Checkin service



(b) satisfaction



(c) Gender

Figure 13: (a) *Checkin service* distribution from original vs generated dataset. (b) *satisfaction* distribution from original vs generated dataset. (c) *Gender* distribution from original vs generated dataset.

Other notable differences arise for *Gender* (see Figure 13b) where females now represent 80,000 individuals compared to 50,000 males, increasing the likelihood of having more females as centroids for the clustering. Additionally, for *satisfaction*, the original dataset has more *neutral or dissatisfied passengers*, whereas this trend is slightly reversed in the synthetic group of passengers, as observed in Figure 13c.

Lastly, for the service-related variables, the category 0 (= Not offered in flight) generally has the fewest occurrences. The model successfully captured this information, although it often results in no occurrences at all in the generation for that modality.

The main profiles resulting from the clustering are displayed in Tables 16, 17 and 18. We immediately note that the variables *Gender* and *Checkin service* mostly showcase their most prominent modality. Moreover, compared to the clustering observed using the Burt matrix and the VAE encoding, we see more predicted satisfaction around the 50%.

The variables *Travel Type* and *Class* still lead to similar conclusions : passengers traveling for personal reasons or in the Eco class are less likely to be satisfied. Again, only the loyal customer type is returned, similar to the two previous cluster analyses, due to the imbalanced distribution of loyal and disloyal customers within the dataset. This is also the case for the flight distance (0-1000) and the departure and arrival delays (0-5).

As a comparison metric, we computed the mean cross-entropy between the predicted satisfaction using the GLM and the satisfaction predicted using the clustering for each synthetic passenger. The mean cross-entropy is equal to 0.4667, which is relatively close to the performance using the Burt matrix.

	Gender	Customer Type	Travel Type	Class	Wifi	Time Conv.	Online booking	Gate	Satisfaction
1	Female	Loyal	Personal	Eco	3	5	3	3	0.0821
2	Female	Loyal	Personal	Eco	2	2	2	2	0.1036
3	Female	Loyal	Business	Business	2	1	2	4	0.1918
4	Female	Loyal	Business	Eco	1	5	2	4	0.2045
5	Female	Loyal	Personal	Eco	4	5	3	3	0.5589
6	Female	Loyal	Business	Business	3	5	3	5	0.5776
7	Male	Loyal	Business	Eco	4	4	3	4	0.6825
8	Female	Loyal	Business	Business	2	2	2	2	0.7994
9	Female	Loyal	Business	Business	4	4	4	4	0.986
10	Female	Loyal	Business	Business	5	5	4	5	0.9945

Table 16: Main profiles, generated data (part 1/3)

	F&D	Online boarding	Seat Comfort	Entertainment	Onboard service	Leg room	Baggage handling	Satisfaction
1	2	3	3	3	3	3	3	0.0821
2	2	2	3	3	3	3	3	0.1036
3	2	2	2	2	2	2	3	0.1918
4	1	1	1	1	3	3	3	0.2045
5	5	4	5	5	5	3	5	0.5589
6	3	4	3	3	3	3	3	0.5776
7	4	4	4	4	4	3	4	0.6825
8	5	2	5	4	4	4	4	0.7994
9	4	4	5	4	4	4	4	0.986
10	5	4	5	5	5	5	5	0.9945

Table 17: Main profiles, generated data (part 2/3)

	Check-in service	Inflight service	Cleanliness	Age	Flight distance	Departure delay	Arrival delay	Satisfaction
1	3	4	3	51-64	0-1000	0-5	0-5	0.0821
2	3	4	3	51-64	0-1000	0-5	0-5	0.1036
3	3	2	2	19-30	0-1000	0-5	0-5	0.1918
4	3	5	1	19-30	0-1000	0-5	0-5	0.2045
5	3	5	5	19-30	0-1000	0-5	0-5	0.5589
6	3	4	3	19-30	0-1000	0-5	0-5	0.5776
7	3	4	4	19-30	0-1000	0-5	0-5	0.6825
8	3	4	5	19-30	0-1000	0-5	0-5	0.7994
9	3	4	4	41-50	0-1000	0-5	0-5	0.986
10	3	5	5	41-50	0-1000	0-5	0-5	0.9945

Table 18: Main profiles, generated data (part 3/3)

6 Conclusion

In this thesis, we explored the potential of applying variational autoencoders (VAEs) to real-world data, specifically concentrating on the satisfaction of passengers of a US airline. We employed the model for various tasks, with a primary focus on clustering to identify main customer profiles. Additionally, we discussed regression and data generation.

The aim was to evaluate VAEs' capabilities in performing dimensionality reduction and learning meaningful latent variables that can be utilized for other tasks affected by the curse of dimensionality. Moreover, given that the VAE's known generative abilities, it was valuable to explore its performance in generating data and assess the quality of this generated data.

Table 19 ranks the five techniques discussed by mean cross-entropy.

Model	Mean cross-entropy
GLM on one-hot encoded data	0.2014
GLM on VAE-encoded data	0.2965
K-means on VAE-encoded data	0.4081
K-means on Burt encoded data	0.4559
K-means on generated data	0.4667

Table 19: Ranking of the techniques seen

The best-performing model in terms of predicting the satisfaction is the generalized linear model applied on the one-hot encoded dataset, achieving a mean cross-entropy of 0.2014. This is followed by the regression on the dataset reduced using the variational autoencoder, which scored 0.2965. Clustering ranks lower with a mean cross-entropy of 0.4081 with the VAE-encoding and 0.4559 with Burt matrix. However, as discussed in section 5.4, increasing the number of clusters could improve the predictive performance, to a point where clustering outperforms regression. Nonetheless, using hundreds of clusters is computationally expensive and often impractical.

Despite this, clustering offers relevant insights into the main profiles of passengers. This technique is for example useful for business and marketing purposes in the context of airline passengers, as it allows to have a glance at the main profiles of customers. Strategies for increasing satisfaction can be implemented by targeting profiles that show the lowest satisfaction.

Additionally, synthetic data was generated using the VAE's generative neural network. Although some variables, such as *Gender* and *Checkin service*, showed imbalances, the clustering still revealed useful patterns, particularly regarding *Travel type* and *Class*. Generating synthetic data keeps on gaining attention as regulations to preserve consumer privacy become more strict. Also, in domains with missing data, data generation is useful for improving research. Analyzing synthetic data does not violate regulations, making it crucial to improve models and explore generative capabilities for performing further analyses.

However, some limitations should be noted. First, the data encoding is suboptimal. We used categorical variables and one-hot encoding, but binary cross-entropy does not connect related levels, allowing an observation to belong to multiple categories.

Moreover, computational resource limitations prevented further model optimization. With more powerful computing resources, it would be possible to optimize models even more through a more extensive grid search.

For future research, model improvements should be explored to enhance performance and improve disentanglement of the latent variables.. For example, adapting the encoding by using a mixed encoding approach, as proposed by Jamotton & Hainaut (2023) could improve data generation.

Additionally, we used an ideal Gaussian prior for data generation. Many other options exist, which could generate data closer to the original set.

Furthermore, working with a variant of the VAE more suitable for clustering could yield interesting results. For instance, Variational Deep Embedding (Jian et al., 2016) and Gaussian Mixture VAE (Dilokthanakul et al., 2016) are more suited for clustering. Applying these techniques to tabular data could be beneficial

Overall, we demonstrated that combining the VAE with clustering effectively models and predicts passenger satisfaction, offering valuable insights into customer profiles. Additionally, regression with one-hot encoded data outperformed mapped data, though both yielded satisfactory results. The VAE also shows potential for creating realistic synthetic datasets for customer satisfaction analysis.

References

Aldunate, Á., Maldonado, S., Vairetti, C., & Armelini, G. (2022). Understanding customer satisfaction via deep learning and natural language processing. Expert Systems With Applications, 209, 118309. https://doi.org/10.1016/j.eswa.2022.118309

An, M., Noh, Y. (2009). Airline customer satisfaction and loyalty : impact of in-flight service quality. Service Business, 3(3), 293-307. https://doi.org/10.1007/s11628-009-0068-4

Asperti, A., Evangelista, D., & Piccolomini, E. L. (2021). A survey on Variational Autoencoders from a GreenAI perspective. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2103.01071

Bank, D., Koenigstein, N., & Giryes, R. (2023). Autoencoders. In Springer eBooks (pp. 353–374). https://doi.org/10.1007/978-3-031-24628-9_16

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2015). Generating Sentences from a Continuous Space. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1511.06349

Denuit, M., Hainaut, D., & Trufin, J. (2019a). Effective Statistical Learning Methods for Actuaries I. In *Springer Actuarial*. https://doi.org/10.1007/978-3-030-25820-7

Denuit, M., Hainaut, D., & Trufin, J. (2019b). Effective Statistical Learning Methods for Actuaries III. In *Springer Actuarial*. https://doi.org/10.1007/978-3-030-25827-6

Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., & Shanahan, M. (2016). Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1611.02648

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1406.2661

Guo, C., & Berkhahn, F. (2016). Entity Embeddings of Categorical Variables. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1604.06737

Hainaut, D. (2018). A self-organizing predictive map for non-life insurance. European Actuarial Journal, 9(1), 173-207. https://doi.org/10.1007/s13385-018-0189-z

Hainaut, D. (2023). Data science for insurance and finance (LDATS2310) [PowerPoint slides]. Université Catholique de Louvain.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2022). beta-VAE : Learning Basic Visual Concepts with a Constrained Variational Framework. OpenReview. https://openreview.net/forum?id=Sy2fzU9gl

Jamotton, C., Hainaut, D. (2023). Variational autoencoder for synthetic insurance data. http://hdl.handle.net/2078.1/276128

Jiang, Z., Zheng, Y., Tan, H., Tang, B., & Zhou, H. (2016). Variational Deep Embedding : An Unsupervised and Generative Approach to Clustering. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1611.05148

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning : Trends, perspectives, and prospects. Science, 349(6245), 255-260. https://doi.org/10.1126/science.aaa8415

Kingma, D. P., & Welling, M. (2013). Auto-Encoding variational Bayes. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1312.6114

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improving Variational Inference with Inverse Autoregressive Flow. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1606.04934

Kingma, D. P., & Welling, M. (2019). An Introduction to Variational Autoencoders. Foundations And Trends In Machine Learning, 12(4), 307-392. https://doi.org/10.1561/2200000056

Larsen, A. B. L., Sønderby, S. K., Larochelle, H., & Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1512.09300

Legrand, C. (2021). Modèles Linéaires Avancés (LSTAT2110) [PowerPoint slides]. Université Catholique de Louvain.

Li, P., Pei, Y., & Li, J. (2023). A comprehensive survey on design and application of autoencoder in deep learning. Applied Soft Computing, 138, 110176. https://doi.org/10.1016/j.asoc.2023.110176

Park, S., Kim, M., Kim, Y., & Park, Y. (2022). A Deep Learning Approach to Analyze Airline Customer Propensities : The Case of South Korea. Applied Sciences, 12(4), 1916. https://doi.org/10.3390/app12041916

Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., & Carin, L. (2016). Variational Autoencoder for Deep Learning of Images, Labels and Captions. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1609.08976

Rezende, D. J., Mohamed, S., Danihelka, I., Gregor, K., & Wierstra, D. (2016). One-Shot Generalization in Deep Generative Models. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1603.05106

Rezende, D. J., & Mohamed, S. (2015). Variational Inference with Normalizing Flows. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1505.05770

Riorini, S. V., & Widayati, C. C. (2018). Customer satisfaction low cost carrier : stimulus and its consequences. Jurnal Manajemen - Fakultas Ekonomi Universitas Tarumanagara/Jurnal Manajemen, 22(1). https://doi.org/10.24912/jm.v22i1.318

Salim, A. (2018). Synthetic Patient Generation : a deep learning approach using variational autoencoders. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1808.06444

Singh, A. & Ogunfunmi, T. (2021). An Overview of Variational Autoencoders for Source Separation, Finance, and Bio-Signal Applications. Entropy, 24(1), 55. https://doi.org/10.3390/e24010055

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., & Winther, O. (2016a). Ladder variational autoencoders. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1602.02282

Song, C., Liu, F., Huang, Y., Wang, L., & Tan, T. (2013). Auto-encoder based data clustering. In Lecture notes in computer science (p. 117-124). https://doi.org/10.1007/978-3-642-41822-8_15

Van Den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural Discrete representation learning. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1711.00937

Van Der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. http://jmlr.org/papers/v9/vandermaaten08a.html

Wei, R., Garcia, C., El-Sayed, A., Peterson, V., & Mahmood, A. (2020). Variations in Variational Autoencoders - A Comparative Evaluation. IEEE Access, 8, 153651-153670. https://doi.org/10.1109/access.2020.3018151

Zaki, M. J., & Meira, W., Jr. (2020). Data Mining and Machine Learning : Fundamental Concepts and Algorithms. Cambridge University Press.

Zhao, S., Song, J., & Ermon, S. (2017). INFOVAE : Information Maximizing Variational Autoencoders. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1706.02262

Airline Passenger Satisfaction. Kaggle. https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction. Downloaded on the 24th of October 2023.

Appendix

6.1 Barplots of the variables

Figure 14: Barplots of the variables in function of satisfaction (part 1/2)

Figure 15: Barplots of the variables in function of satisfaction (part 2/2)

Figure 16: Barplots of the variables (part 1/2)

Figure 17: Barplots of the variables (part 2/2)

6.2 Correlation matrix of latent z

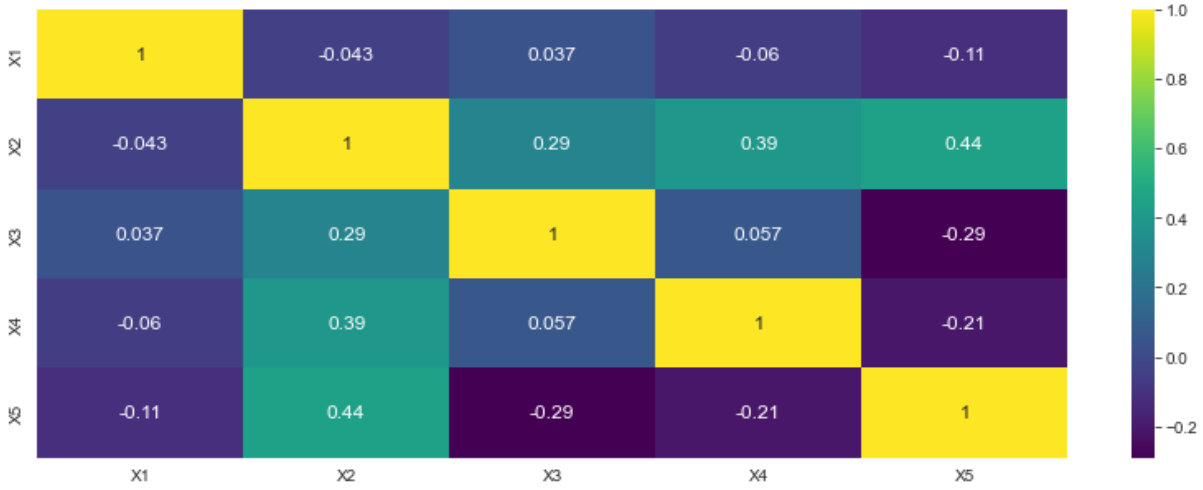


Figure 18: Correlation matrix of latent z using VAE adapted for K-means

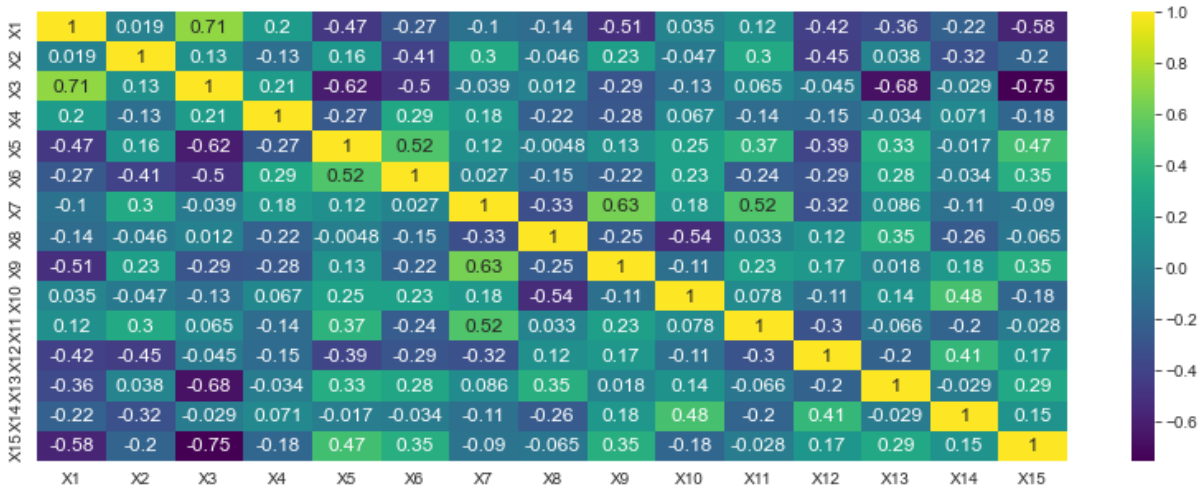


Figure 19: Correlation matrix of latent z using VAE adapted for GLM

6.3 Results of the GLM on the encoded dataset

Generalized Linear Model Regression Results

Dep. Variable:	satisfaction	No. Observations:	103904
Model:	GLM	Df Residuals:	103889
Model Family:	Binomial	Df Model:	14
Link Function:	Logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-30882.
Deviance:	61765.	Pearson chi2:	1.09e+05
No. Iterations:	7	Pseudo R-squ. (CS):	0.5391
Covariance Type:	nonrobust		

Table 20: Model specifications of the GLM with encoded data

	coef	std err	z	$P > \|z\|$	[0.025	0.975]
x1	0.5124	0.066	7.732	0.000	0.383	0.642
x2	0.2970	0.028	10.575	0.000	0.242	0.352
x3	-0.5033	0.043	-11.737	0.000	-0.587	-0.419
x4	1.0620	0.030	35.279	0.000	1.003	1.121
x5	-0.5237	0.055	-9.543	0.000	-0.631	-0.416
x6	-0.6798	0.054	-12.522	0.000	-0.786	-0.573
x7	-0.8746	0.047	-18.587	0.000	-0.967	-0.782
x8	0.8273	0.054	15.224	0.000	0.721	0.934
x9	0.6402	0.053	12.108	0.000	0.537	0.744
x10	0.5924	0.061	9.767	0.000	0.474	0.711
x11	-0.1562	0.046	-3.399	0.001	-0.246	-0.066
x12	-0.3786	0.067	-5.668	0.000	-0.510	-0.248
x13	-0.5950	0.044	-13.392	0.000	-0.682	-0.508
x14	-0.2073	0.030	-6.987	0.000	-0.265	-0.149
x15	-0.8296	0.029	-28.200	0.000	-0.887	-0.772

Table 21: Coefficients of GLM with encoded variables

6.4 Results of the GLM on the one-hot encoded dataset

Generalized Linear Model Regression Results

Dep. Variable:	satisfaction	No. Observations:	103904
Model:	GLM	Df Residuals:	103820
Model Family:	Binomial	Df Model:	83
Link Function:	Logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-20638.
Deviance:	41276.	Pearson chi2:	3.04e+05
No. Iterations:	8	Pseudo R-squ. (CS):	0.6216
Covariance Type:	nonrobust		

Table 22: Model specifications of the GLM with one-hot encoded data

variable	coef	std err	z	$P > \|z\|$	[0.025	0.975]
const	1.2110	0.454	2.666	0.008	0.321	2.101
Gender_Male	0.2970	0.026	1.614	0.107	-0.009	0.092
Customer Type_disloyal Customer	-3.0174	0.046	-66.153	0.000	-3.107	-2.928
Type of Travel_Personal Travel	-3.9572	0.050	-78.890	0.000	-4.055	-3.859
Class_Eco	-0.6273	0.035	-17.875	0.000	-0.696	-0.559
Class_Eco Plus	-0.7888	0.056	-14.029	0.000	-0.899	-0.679
Inflight wifi service_2	-0.9526	0.057	-16.705	0.000	-1.064	-0.841
Inflight wifi service_3	-0.9185	0.057	-16.138	0.000	-1.030	-0.807
Inflight wifi service_4	0.6795	0.055	12.420	0.000	0.572	0.787
Inflight wifi service_5	5.7897	0.127	45.722	0.000	5.541	6.038
Departure/Arrival time convenient_1	0.5530	0.081	6.865	0.000	0.395	0.711
Departure/Arrival time convenient_2	0.8439	0.078	10.801	0.000	0.691	0.997
Departure/Arrival time convenient_3	0.6677	0.075	8.876	0.000	0.520	0.815
Departure/Arrival time convenient_4	-0.2976	0.066	-4.479	0.000	-0.428	-0.167
Departure/Arrival time convenient_5	-0.4881	0.072	-6.740	0.000	-0.630	-0.346
Ease of Online booking_1	-3.7499	0.153	-24.462	0.000	-4.050	-3.449
Ease of Online booking_2	-3.7500	0.155	-24.181	0.000	-4.054	-3.446
Ease of Online booking_3	-3.2808	0.153	-21.415	0.000	-3.581	-2.981
Ease of Online booking_4	-2.5332	0.151	-16.791	0.000	-2.829	-2.237
Ease of Online booking_5	-3.1457	0.156	-20.171	0.000	-3.451	-2.840

Table 23: Coefficients of GLM with one-hot encoded variables (part 1/3)

variable	coef	std err	z	$P > \|z\|$	[0.025	0.975]
Gate location_2	0.1035	0.055	1.890	0.059	-0.004	0.211
Gate location_3	-0.1259	0.051	-2.487	0.013	-0.225	-0.027
Gate location_4	-0.2884	0.052	-5.540	0.000	-0.390	-0.186
Gate location_5	-0.4779	0.068	-6.991	0.000	-0.612	-0.344
Food and drink_1	0.0801	0.333	0.240	0.810	-0.573	0.733
Food and drink_2	0.3838	0.332	1.155	0.248	-0.268	1.035
Food and drink_3	0.2528	0.332	0.762	0.446	-0.397	0.903
Food and drink_4	0.2857	0.331	0.862	0.388	-0.364	0.935
Food and drink_5	0.1184	0.332	0.356	0.722	-0.533	0.770
Online boarding_1	-0.9612	0.157	-6.132	0.000	-1.268	-0.654
Online boarding_2	-0.7094	0.157	-4.526	0.000	-1.017	-0.402
Online boarding_3	-0.9634	0.156	-6.192	0.000	-1.268	-0.658
Online boarding_4	0.6434	0.155	4.155	0.000	0.340	0.947
Online boarding_5	1.8584	0.159	11.721	0.000	1.548	2.169
Seat comfort_2	-0.4206	0.068	-6.165	0.000	-0.554	-0.287
Seat comfort_3	-1.5271	0.063	-24.050	0.000	-1.652	-1.403
Seat comfort_4	-0.8513	0.063	-13.534	0.000	-0.975	-0.728
Seat comfort_5	-0.1015	0.068	-1.496	0.135	-0.234	0.031
Inflight entertainment_2	0.7345	0.091	8.105	0.000	0.557	0.912
Inflight entertainment_3	1.5789	0.084	18.724	0.000	1.414	1.744
Inflight entertainment_4	1.2943	0.079	16.285	0.000	1.138	1.450
Inflight entertainment_5	0.5673	0.089	6.410	0.000	0.394	0.741
On-board service_2	0.1160	0.061	1.897	0.058	-0.004	0.236
On-board service_3	0.5614	0.055	10.220	0.000	0.454	0.669
On-board service_4	0.6485	0.055	11.799	0.000	0.541	0.756
On-board service_5	1.1341	0.060	18.812	0.000	1.016	1.252
Leg room service_1	0.6102	0.195	3.126	0.002	0.228	0.993
Leg room service_2	0.9595	0.194	4.941	0.000	0.579	1.340
Leg room service_3	0.8323	0.194	4.286	0.000	0.452	1.213
Leg room service_4	1.4907	0.194	7.682	0.000	1.110	1.871
Leg room service_5	1.6716	0.195	8.579	0.000	1.290	2.053
Baggage handling_2	-0.0672	0.071	-0.951	0.342	-0.206	0.071
Baggage handling_3	-0.6477	0.066	-9.858	0.000	-0.776	-0.519
Baggage handling_4	-0.0645	0.064	-1.007	0.314	-0.190	0.061
Baggage handling_5	0.5791	0.068	8.521	0.000	0.446	0.712

Table 24: Coefficients of GLM with one-hot encoded variables (part 2/3)

variable	coef	std err	z	$P > \|z\|$	[0.025	0.975]
Checkin service_2	0.1748	0.050	3.498	0.000	0.077	0.273
Checkin service_3	0.7026	0.045	15.750	0.000	0.615	0.790
Checkin service_4	0.6452	0.044	14.504	0.000	0.558	0.732
Checkin service_5	1.3634	0.051	26.721	0.000	1.263	1.463
Inflight service_2	-0.1552	0.075	-2.076	0.038	-0.302	-0.009
Inflight service_3	-0.8542	0.069	-12.298	0.000	-0.990	-0.718
Inflight service_4	-0.2298	0.067	-3.453	0.001	-0.360	-0.099
Inflight service_5	0.4256	0.071	6.021	0.000	0.287	0.564
Cleanliness_2	-0.0347	0.068	-0.511	0.609	-0.168	0.098
Cleanliness_3	0.4357	0.062	7.083	0.000	0.315	0.556
Cleanliness_4	0.3083	0.062	5.011	0.000	0.188	0.429
Cleanliness_5	0.8198	0.072	11.435	0.000	0.679	0.960
AgeCat_19 - 30	0.2646	0.057	4.606	0.000	0.152	0.377
AgeCat_31 - 40	-0.0205	0.058	-0.351	0.726	-0.135	0.094
AgeCat_41 - 50	0.3557	0.059	6.053	0.000	0.241	0.471
AgeCat_51 - 64	0.1930	0.059	3.246	0.001	0.076	0.310
AgeCat_65-85	-0.4738	0.082	-5.763	0.000	-0.635	-0.313
FlightDistanceCat_1001 - 2000	0.0921	0.034	2.739	0.006	0.026	0.158
FlightDistanceCat_2001 - 3000	0.0035	0.042	0.081	0.935	-0.080	0.087
FlightDistanceCat_3001 - 4000	0.0539	0.052	1.037	0.300	-0.048	0.156
FlightDistanceCat_4001 - 5000	1.3771	0.442	3.114	0.002	0.510	2.244
DepartDelayCat_6 - 60	0.0338	0.036	0.942	0.346	-0.037	0.104
DepartDelayCat_61 - 120	0.0998	0.107	0.934	0.350	-0.110	0.309
DepartDelayCat_121 - 240	0.3054	0.197	1.553	0.120	-0.080	0.691
DepartDelayCat_240+	-0.4824	0.451	-1.070	0.285	-1.366	0.401
ArrivalDelayCat_6 - 60	-0.7198	0.036	-20.068	0.000	-0.790	-0.649
ArrivalDelayCat_61 - 120	-0.7844	0.106	-7.372	0.000	-0.993	-0.576
ArrivalDelayCat_121 - 240	-0.9547	0.193	-4.939	0.000	-1.334	-0.576
ArrivalDelayCat_241+	-0.5702	0.438	-1.303	0.193	-1.428	0.288

Table 25: Coefficients of GLM with one-hot encoded variables (part 3/3)

6.5 Barplots of the variables of the generated data

Figure 20: Barplots of the variables of generated dataset (part 1/2)

Figure 21: Barplots of the variables of generated dataset (part 2/2)

6.6 Code resources

https://github.com/chlomarchal/ClusteringWithVAE