Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jaka Klančar

Generatorji delno umetnih podatkov na podlagi samokodirnikov

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Marko Robnik Šikonja

Ljubljana, 2018

University of Ljubljana
Faculty of Computer and Information Science

Jaka Klančar

Autoencoder based generators of semi-artificial data

master's thesis

the 2nd cycle master's study programme
computer and information science

Supervisor: Prof. Marko Robnik Šikonja, Ph.D.

Ljubljana, 2018

Acknowledgments

I would first like to thank my mentor Prof. Marko Robnik Šikonja. He was always ready to help whenever I had a question about my research or writing. He consistently allowed this thesis to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to acknowledge Jasmine Brown as proofreader of this thesis. I am thankful for her valuable comments.

Finally, I must express my very profound gratitude to my parents, my siblings and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. I would especially like to thank my brother, as he was one of the main reasons I decided to study computer science.

This accomplishment would not have been possible without them. Thank you.

Jaka Klančar, 2018

Contents

Povzetek

Naslov: Generatorji delno umetnih podatkov na podlagi samokodirnikov

Glavni cilj naloge je bil olajšati problem pomanjkanja podatkov pri analizi podatkov in v strojnem učenju. Razvili smo generator delno umetnih podatkov na podlagi samokodirnikov. Implementirali smo dinamične samokodirnike brez vnaprej določene strukture, saj smo želeli, da so generatorji uporabni na poljubni učni množici. Rezultati so pokazali, da generatorji na podlagi samokodirnikov delujejo bolje kot variacijski samokodirniki. Naši generatorji najbolje delujejo na podatkovnih množicah z manjšim številom atributov in z uravnoteženimi razredi. Večje število učnih primerov izboljša delovanje generatorjev. Rezultati so tudi pokazali, da z mrežnim iskanjem znatno izboljšamo rezultate in da je možno napovedati dobre parametre glede na karakteristike dane podatkovne množice.

Ključne besede

samokodirniki, generatorji podatkov, nevronske mreže, delno umetni podatki, variacijski samokodirniki

Abstract

Title: Autoencoder based generators of semi-artificial data

The goal of the thesis is to alleviate the problem of insufficient data available for data analysis or machine learning. We developed a generator of semi-artificial data based on autoencoders. We implemented dynamic autoencoders without any predefined structure, as we wanted that our solution is general and may therefore be used on any data set. Results showed that autoencoder based generators work better than variational autoencoders. The generators perform best on data sets with a small number of mixed attributes and balanced classes. They perform better if more training instances are available. Results additionally show that grid search significantly improves the performance and that it is possible to predict a good set of parameters for each data set.

Keywords

autoencoders, data generators, neural networks, semi-artifical data, variational autoencoders

Razširjeni povzetek

Kljub temu, da je dandanes veliko govora o problemu velepodatkov, obstaja mnogo problemov, kjer nimamo dovolj podatkov za analizo oziroma za strojno učenje. Kot primer lahko vzamemo problem detekcije redkih bolezni. Razlogi za pomanjkanje podatkov so težave pri zbiranju podatkov (npr. zasebnost podatkov), visoka cena (npr. draga oprema), redkost podatkov (npr. redke bolezni) ali neuravnotežena distribucija razredov (npr. zloraba kreditnih kartic).

Problem primanjkljaja podatkov je bil že obravnavan z generatorji na podlagi RBF (angl. *radial basis function*) mrež in naključnih gozdov (angl. *random forests*) [31]. Ta rešitev ima nekaj pomanjkljivosti, še posebej, ko imamo veliko odvisnosti med atributi. Naš cilj je bil preseči te rezultate z generatorji na podlagi samokodirnikov (angl. *autoencoders*).

I Pregled sorodnih del

Robnik-Šikonja [31] je predstavil idejo generiranja delno umetnih podatkov z RBF mrežami. Uporabnost generatorja je bila ovrednotena na 51 podatkovnih množicah. Rezultati so pokazali, da so originalni in generirani podatki precej podobni in da je ta metoda lahko uporabna pri precejšnjem številu podatkovnih množic.

Podoben primer je raziskoval Miranda et al [27]. Poskušali so rekonstruirati manjkajoče podatke z uporabo samokodirnikov. V članku je opisana metoda za rekonstrukcijo v primeru nepričakovane ustavitve sistema, pri čemer

lahko pride do izgube pomembnih podatkov.

Li, Loung and Jurafsky [24] so predstavili uporabo samokodirnikov pri obdelavi naravnega jezika. Raziskovali so možnost generiranja daljših odstavkov. Lu et al [25] je uporabil samokodirnike za zmanjševanje šuma in izboljšanje govora. Na vhod samokodirnikov so dali govor s šumom, na izhodu so pa poskušali dobiti govor brez šuma. Pokazali so, da ta pristop deluje, vendar le za veliko učno množico. Bengio [4] je podrobneje opisal različne globoke nevronske mreže, vključno s samokodirniki. Pokazal je motivacijo uporabe algoritmov za globoko učenje in predstavil, kje se ti algoritmi uporabljajo.

II Generatorji delno umetnih podatkov na podlagi samokodirnikov

Glavni cilj te magistrske naloge je bil razviti delujoče generatorje podatkov z uporabo samokodirnikov. Med razvojem smo se odločili, da bomo dodatno implementirali še variacijske samokodirnike (angl. *variational autoencoders*).

II.I Samokodirniki

Ideja dinamičnih samokodirnikov je preprosta, saj nevronski mreži dodajamo skrite nivoje dokler se rezultati izboljšujejo. V našem primeru izboljševanje rezultata pomeni, da je seštevek kriterijske funkcije in nekega praga manjši od izgube v prejšnji iteraciji. Algoritem 1 v poglavju 3.2.1 prikazuje psevdokodo naše rešitve. Na začetku imamo le vhodni nivo, izhodni nivo in en skriti nivo. V vsakem koraku nato dodamo nov srednji nivo, katerega velikost je odvisna od parametra `r` in se izračuna kot $new\_size = prev\_size * r$. Prejšnji srednji nivo postane zadnji nivo v kodirnem delu samokodirnika in prvi nivo v dekodirnem delu samokodirnika.

II.II Variacijski samokodirniki

Variacijski samokodirniki imajo podobno rešitev kot navadni. Glavna razlika je v srednjem nivoju, kjer imamo sedaj namesto enega kar dva nivoja. V prvem izmed teh dveh nivojev imamo parametra z_mean in z_log_var, ki modelirata normalno porazdelitev in predstavljata latenten prostor vhodnih podatkov. Drugi nivo vzorči podatke iz latentne normalne porazdelitve $z = z\_mean + exp(z\_log\_var) * \mathcal{N}(0, 1)$.

III Evaluacija

Generirane podatke smo testirali in primerjali z originalnimi podatki. Podatki bi morali imeti podobno strukturo, podobne statistične lastnosti in vračati podobne rezultate pri metodah strojnega učenja. Generatorji so bili testirani na 52 podatkovnih množicah. Testiranje smo izvedli s 5 x 10 prečnim preverjanjem. V vsaki iteraciji smo generirali 1000 novih primerov podatkov. Podatke smo primerjali glede na statistične lastnosti, gručenje in klasifikacijsko točnost pri uporabi naključnih gozdov.

Statistične lastnosti, ki smo jih poročali so:

- razlika v povprečju m($\Delta mean$),
- razlika v standardnem odklonu m(Δstd),
- razlika v koeficientu simetrije m($\Delta\gamma_1$),
- razlika v sploščenosti m($\Delta\gamma_2$).

Gručenje smo izvedli za originalne učne podatke in generirane podatke. Nato smo originalnim testnim podatkom priredili gručo glede na najbližjega soseda za obe gručenji. Izračunali smo *Adjusted Rand index* (**ARI**), ki nam pove kako podobni sta dve gručenji. Ko je ARI enak 1, pomeni da sta gručenji identični. Vrednost 0 pomeni, da sta gručenji popolnoma naključni.

Pri izvedbi klasifikacije smo poročali vrednosti **m1d1**, ki nam pove točnost klasifikacije testnih podatkov glede na originalne učne podatke, in **m2d1**, ki

nam pove točnost klasifikacije testnih podatkov glede na generirane podatke. Izračunana $\Delta(m_1, m_2) = m2d1 - m1d1$ nam pove razliko med **m1d1** in **m2d1**.

Tabela 1 prikazuje povprečne rezultate za različna testiranja. Testirali smo samokodirnike (okrajšava AE) in variacijske samokodirnike (okrajšava VAE) s privzetimi parametri. Nato smo uporabili 17 podatkovnih množic, na katerih so bili testirani generatorji na podlagi RBF mrež [31] in jih primerjali z našimi generatorji.

Generatorje smo testirali z mrežnim iskanjem (angl. *grid search*). Poskusili smo tudi napovedati optimalno kombinacijo parametrov glede na rezultate mrežnega iskanje.

	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
AE s privzetimi parametri	0.022	0.041	0.46	3.42	0.73	72.18	62.62	−9.55
VAE s privzetimi parametri	0.124	0.107	1.21	5.34	0.43	71.63	48.47	−23.16
AE na pod. množicah iz [31]	0.015	0.026	0.18	0.81	0.62	78.66	61.89	−16.78
RBF na pod. množicah iz [31]	0.027	0.019	0.20	0.58	0.58	77.84	72.59	−5.25
AE z mrežnim iskajem	0.030	0.067	0.70	3.57	0.60	63.4	59.8	−3.5
AE z napovedovanjem komb. parametrov	0.015	0.035	0.19	0.73	0.59	79.00	64.18	−14.82

Tabela 1: Rezultati testiranj.

IV Sklep

Rezultati kažejo, da generatorji, ki temeljijo na samokodirnikih delujejo bolje, kot generatorji, ki temeljijo na variacijskih samokodirnikih. Kljub temu so naši generatorji v večini primerov slabši kot generatorji na podlagi RBF mrež [31]. Menimo, da naši generatorji najbolje delujejo na podatkovnih množicah z manjšim številom atributov in uravnoteženimi razredi. Večje število učnih primerov izboljša delovanje generatorjev.

Glavni problem naših generatorjev je nastavitev parametrov, ki bodo vrnili najboljše rezultate za določeno podatkovno množico. Privzete parametre smo nastavili s testiranjem posameznega parametra neodvisno od drugih. Ta pristop ima precej pomanjkljivosti, saj so parametri odvisni med sabo in od

podatkovne množice. Pokazali smo, da lahko rezultate znatno izboljšamo z uporabo mrežnega iskanja, ki pa je računsko zelo zahtevno in posledično zamudno. V poskusu, da bi se izognili iskanju najboljše kombinacije parametrov z mrežnim iskanjem, smo implementirali sistem za napovedovanje dobre kombinacije parametrov. Ta sistem vrača približno enako dobre rezultate kot privzeti parametri. Menimo, da bi se rezultati precej izboljšali, če bi imeli večje število učnih podatkov pri učenju modela za napovedovanje.

Implementirani generatorji niso primerni za sekvenčne podatke ali slike. Da bi lahko uporabljali tudi te tipe podatkov, bi morali razviti nova algoritma, samokodirnike na podlagi LSTM za sekvenčne podatke in konvolucijske samokodirnike za slike [9].

Chapter 1

Introduction

Even though big data is a hot topic nowadays, there are many problems, where there is not enough data available for data analysis or machine learning. An example is detecting or analysing rare diseases. The reasons for insufficient data are difficulties in obtaining data (privacy of data), high cost (expensive equipment), rarity of data (rare diseases) or imbalanced distribution of events (credit card fraud detection). To tackle this problem, we will develop a semi-artificial data generator. The same problem has already been addressed with generators based on radial basis function (RBF) networks and random forests [31]. These solutions have certain shortcomings, especially when dealing with many dependencies among attributes. Our aim is to improve upon existing approach.

The lack of data in machine learning causes problems in model selection, performance estimation, development of specialized algorithms and tuning of learning model parameters. Certain problems caused by scarce data are inherent to underrepresentation of the problem and cannot be solved, but some aspects can be eased by generating artificial data similar to the original one. Similar artificial data sets can help in tuning the parameters, development of specialized solutions, simulations, and imbalanced problems, as they prevent overfitting of the original data set. If we do not have any background knowledge about the problem, we have to use the data available to extract some

of its properties and generate new semi-artificial data with similar properties. We assume that we can afford to take at least a small part of the data for generating new data. As the proposed generator is a general solution, it is up to developers to decide whether using it is acceptable for a given problem [31].

This thesis is organized as follows. In Section 2, we review the theory behind autoencoders and research about related works, where the problem of generating semi-artificial data was addressed. In Section 3, we present two implemented solutions, dynamic autoencoders and dynamic variational autoencoders. We explain details on preprocessing and generating data. In Section 4, we present the data sets used for evaluation and explain how the performance of our solution was evaluated. In Section 5, we analyze the quality of the generated data. We analyze strengths and weaknesses of our proposed generators. In Section 6, we conclude with a summary, analysis, and ideas for possible improvements.

Chapter 2

Artificial Neural Networks

Machine learning is a process, which helps us to detect patterns in data. Classification algorithms, used in machine learning, first learn how to detect patterns in training data and can later use that knowledge on new, previously unseen, data. Artificial neural networks are one of successful machine learning algorithms.

Schmidhuber [33] summarizes relevant work in the field of neural networks. A neural network consists of simple, connected units called neurons. Each neuron receives one or more inputs and sums them to produce an output. Usually, each input is separately weighted, and the sum is passed through an activation function or transfer function. Finding weights and parameters that cause the neural network to perform a desired behaviour is called learning. Depending on the problem and how the neurons are connected, the learning phase is relatively time-consuming [33].

Neural networks are used as predictors in games, for detection and recognition in computer vision and many other classification problems. According to Tyantov [38] the biggest developments using deep neural networks in the past year (2017) were in the fields of text translation, voice generation, computer vision, reinforcement learning and generating data.

2.1 Autoencoders

Autoencoders, also known as diabolo networks [34], autoassociative neural networks [21] and replicator neural networks [15], are symmetrical neural networks, with the middle layer representing an encoding of the input data [6]. Autoencoders are trained to encode the input x into some representation $c(x)$, from which the input can be reconstructed (see Figure 2.1). The target output of the autoencoder is equal to the input [4]. Autoencoders are rarely used in practical applications, two notable applications are data denoising and dimensionality reduction for data visualization [9].

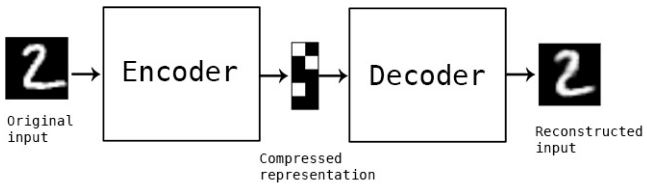


Figure 2.1: Autoencoder structure (Source: [9]).

The structure of an autoencoder is a feedforward neural network. Since the objective is to reproduce the input data on the output layer, both input and output have the same dimension. The encoding, $c(x)$, can be higher or lower dimensional in comparison to the input, depending on the task and desired behaviour. The autoencoders can have many layers, usually placed symmetrically in the encoder and decoder [6]. Charte et al [6] offers a tutorial for development of autoencoders.

2.1.1 Activations Functions

Each unit in hidden layers of a neural network receives inputs from the preceding layer. The unit computes the weighted sum of the inputs and applies a certain operation – the activation function. This function produces the output of the unit. Activation functions used in this thesis are rectified linear units (shortly ReLu), hyperbolic tangent (shortly tanh) and sigmoid

functions.

Sigmoid Sigmoid activation function, also known as logistic function, takes any real number and transforms it to an interval between 0 and 1, see Figure 2.2 and Equation (2.1).

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

The sigmoid function was frequently used in the past. Recently it has been falling out of favour, because of the drawbacks in comparison to other activation functions [17].

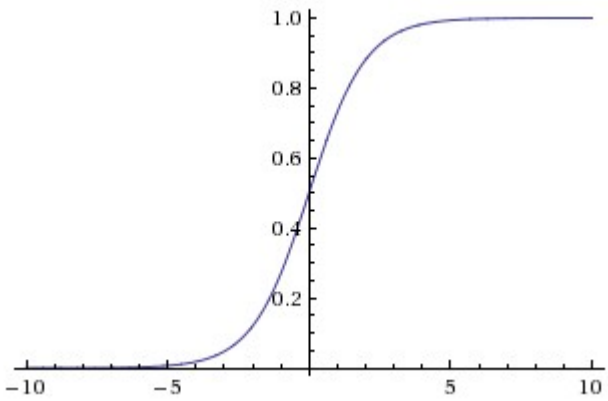


Figure 2.2: Sigmoid activation function (Source: [17]).

tanh The hyperbolic tangent is similar to the sigmoid function, but it transforms any real number to an interval between -1 and 1, see Figure 2.3 and Equation (2.2). In practice, tanh is preferred to the sigmoid function [17].

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{2.2}$$

ReLu The Rectified Linear Unit transform an input using function $f(x) = max(0, x)$, see Figure 2.4 and Equation (2.3). Even thought ReLu is popular

in many deep learning models, it tends to degrade a performance of autoencoders. The main reason is that it always outputs 0 for negative inputs, therefore it weakens the reconstructing process of the autoencoder [6].

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.3}$$

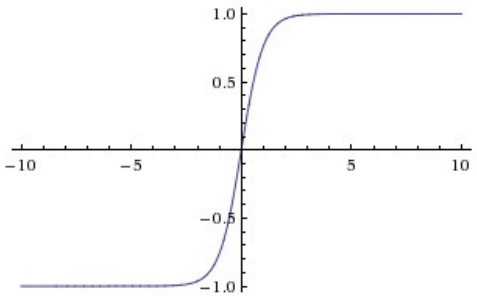


Figure 2.3: tanh activation function (Source: [17]).

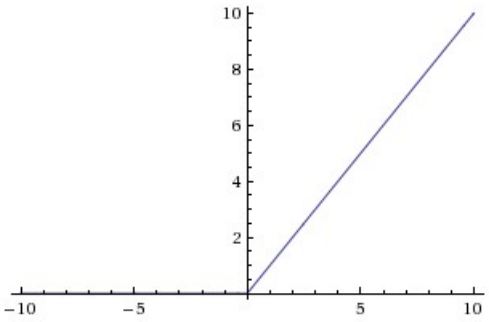


Figure 2.4: ReLu activation function (Source: [17]).

When designing neural networks with multiple hidden layers, it is possible to use different activation functions in different layers. This would result in the network combining the characteristics of several of these functions. Nonetheless, this is rarely used in practice.

2.1.2 Loss Functions

Activation functions used within each layer should be chosen according to the loss function being optimized. Using ReLu at the output can be practical when using mean squared error as the reconstruction error. On the other hand, a sigmoid activation functions combines better with the cross-entropy error and normalized data, since it outputs values between 0 and 1 [6].

2.1.3 Related work

There are many articles describing autoencoders and their use. Li, Loung and Jurafsky [24] show usability of autoencoders in natural language processing. They explore the possibility to generate multi-sentence paragraphs. They show that neural models are able to encode texts and preserve syntactic, semantic and discourse coherence. Lu et al [25] used deep autoencoders for noise reduction and speech enhancement. They pretrained each layer as one hidden layer autoencoder, using noisy speech as an input and clean speech as an output. The results show that the approach improves the performance, but only with a large training dataset.
Bengio [4] offers an insight in different deep architectures, including autoencoders. It discusses motivations and principles regarding learning algorithms for deep architectures.

2.2 Variational Autoencoders

Variational autoencoders (VAEs) became one of the most popular approaches for unsupervised learning of complicated distributions in recent years [11]. The potential of VAEs can be seen in several articles that generate different kinds of complex data, for example faces [20, 30, 22], handwritten digits [20, 32], house numbers [19, 14] and others [36, 39].

The mathematical basis of variational autoencoders has almost nothing in common with classical autoencoders. VAEs are called autoencoders only because the model consists of an encoder and a decoder, which resembles a traditional autoencoder [11].

Variational autoencoders suppose that there exists some hidden variable z which generates an x, where x is input data. To generate new data, similar to the original data, we need to calculate $p(z|x)$:

$$p\left(z|x\right) = \frac{p\left(x|z\right)p\left(z\right)}{p\left(x\right)}$$

As computing $p(x)$ is difficult, $p(z|x)$ can be approximated by another distri-

bution $q(z|x)$, which is defined in a way that makes it a tractable distribution, which means it can be calculated in polynomial time. The goal is to define the parameters of $q(z|x)$ to make it very similar to $p(z|x)$, see Figure 2.5. The Kullback–Leibler divergence is used to measure the difference between two probability distributions. Therefore minimizing Kullback–Leibler divergence causes $q(z|x)$ to be similar to $p(z|x)$ [16].

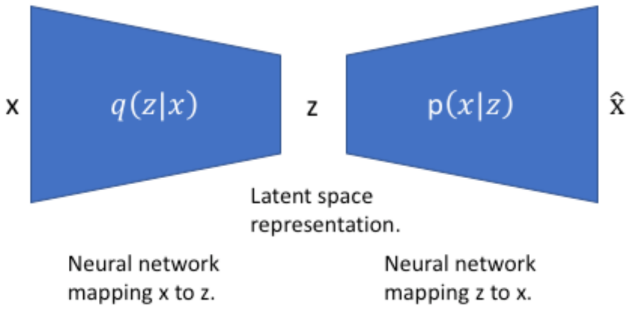


Figure 2.5: Graphic presentation of a statistical theory of a variational autoencoder (Source: [16]).

2.2.1 Implementation

An encoder part of a VAE outputs parameters of distributions of each dimension in the latent space, rather than outputting values for the latent state like a standard autoencoder. Because it is assumed that a prior follows a normal distribution, the encoder outputs two vectors describing the mean and variance of the latent state distributions (see Figure 2.6). To build a multivariate Gaussian model, a covariance matrix would need to be defined to express the correlation of the dimensions. It is assumed that the covariance matrix only has non-zero values on the diagonal to simplify computation [16].

An decoder part of a VAE generates a latent vector by sampling from these defined distributions and proceeds to reconstruct the original input. The decoder model can be used as a generative model. By sampling from

the latent space, it is capable of creating new data similar to a training data [16].

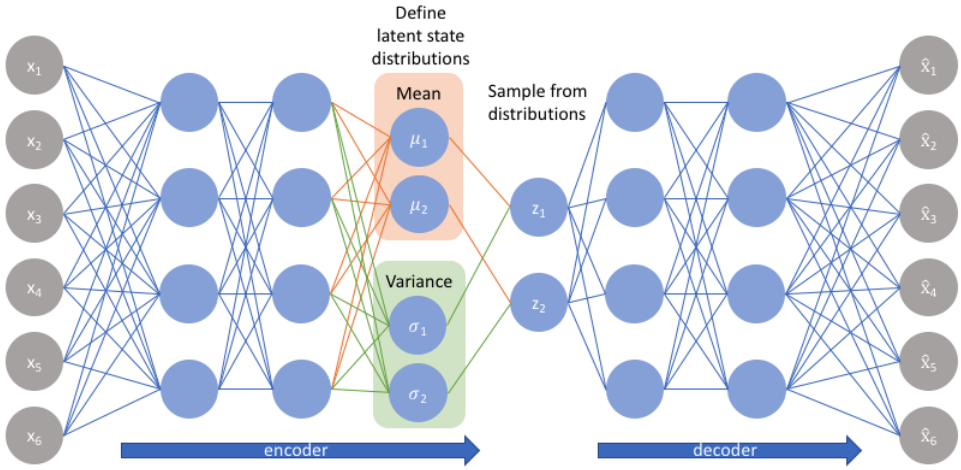


Figure 2.6: Graphic presentation of variational autoencoders (Source: [16]).

A loss function for VAEs consist of two parts, one penalizing reconstruction error (for example, cross-entropy) and a second part, encouraging a learned distribution $q(z|x)$ to be similar to the true prior distribution $p(z)$, see (Equation 2.4). It is assumed that $p(z)$ follows a normal distribution $\mathcal{N}(0,\ 1)$ for each dimension of the latent space [16].

$$\mathcal{L}\left(x, \hat{x}\right) + \beta \sum_{j} KL\left(q_j\left(z|x\right)||N\left(0, 1\right)\right) \tag{2.4}$$

2.3 Data Generators Based on Neural Networks

Robnik-Šikonja [31] presented the idea of generating semi-artificial data using RBF networks. The usability of the proposed generator was evaluated on 51 data sets. The results showed a considerable similarity between the original and generated data. The results suggest that the method could be useful in several scenarios. Slightly similar problem was addressed by Miranda et al [27] who tried to reconstruct missing data using autoencoders. The

article presents a solution for reconstruction of missing information at energy distribution management systems. After unexpected shutdown of the system some crucial information, e.g. missing voltage values, might be missing. The solution performs well in reconstructing missing voltage and power values.

Goodfellow et al. [12] proposes a method for estimating generative models via an adversarial process. Two models are trained simultaneously. A generative model, capturing the data distribution, and a discriminative model, estimating the probability that a sample came from the training data rather than generated data. The training phase tries to optimize the generative model to maximize the probability of the discriminative model making a mistake. Analysis of the generated data shows the potential of the method through qualitative and quantitative evaluation.

Chapter 3

Generating Semi-Artificial Data Using Autoencoders

The main goal of the thesis was to develop a working data generator using autoencoders. During the development, we decided to also implement a solution using variational autoencoders, which could possibly give better results than classical autoencoders. The solution consists of four modules: preprocessing, dynamic model choosing and training, data generating and evaluation.

We used Python libraries to ease the development process. Keras, which provides building blocks for developing deep learning models [8]. Pandas [26] and Numpy [28] were used for reading and processing the data. For evaluating and comparing the generated data with the original data, we used Scikit-learn [29], which offers simple and efficient tools for data mining and data analysis.

3.1 Data Preprocessing

The data needs to be prepared in an appropriate format in order for our model to be able to process it. Missing or not applicable values need to be imputed or dropped. Ideally, the user would deal with those values, as

missing values can have different meaning in different data sets. By default, we separately impute numerical and categorical data. For each numerical attribute, we calculate the mean of the non-missing values and fill the missing values with the mean. For the categorical attributes, we treat missing values as an additional category. Even though better imputation solutions exist, we use these simple techniques due to their efficiency.

Numerical attributes are normalized to $[0, 1]$ interval and categorical attributes are encoded as one-hot vectors, since our libraries expect only numerical data as input. The one-hot encoding encodes a categorical attribute to n binary attributes, where n is number of categories. For example, *Fruit* attribute with three categories would be encoded with three binary attributes F_{banana}, F_{apple} and F_{pear}. So, if the *Fruit* attribute has value $Fruit = Banana$, it is encoded as $F_{banana} = 1$, $F_{apple} = 0$ and $F_{pear} = 0$. The normalization parameters are saved, in order to transform the generated data back to the original form.

3.2 Autoencoders

We have to set an autoencoder's architecture in order to generate new data. As we want our solution to be as general as possible and be available for many data sets, we developed dynamic autoencoders.

3.2.1 Model

The idea behind our dynamic autoencoders is simple – add hidden layers as long as the loss value is improving. In our case, the improvement in the loss value means that the sum of the loss value of the current structure of the autoencoder and the predetermined threshold is smaller than loss value of the previous structure. The threshold is set to 1% of the loss of the previous structure. The threshold is used to make the solution more efficient, as a small improvement is not worth of another step due to learning effort required for additional hidden layer.

The Algorithm 1 shows the pseudocode of the solution. We start with the input layer, the output layer and one hidden (encoding) layer in between, see Figure 3.1. In every step we add a new encoding layer, whose size depends on the `r` parameter. The size of the new encoding layer is geometrically reduced as $new\_size = prev\_size * r$. If `r` is greater than one, the hidden layers have higher dimensions than input layer. Additionally, if the input size is less than a 100, the first hidden layer is twice as big as the input layer (see Figure 3.2). The previous encoding layer becomes the last hidden layer in the encoder and the first hidden layer in decoder.

All the layers, except the output layer use `ReLu` activation function, while the output layer uses sigmoid activation function. Since we normalize the data to the interval $[0, 1]$, using sigmoid activation guarantees that the generated data will also be on the interval $[0, 1]$. The `r` parameter has a default value 0.8. Autoencoder is trained with the `Adam` algorithm [18] for a total of 100 epochs with the default batch size of 16. It uses binary cross-entropy as a loss function. The model uses early stopping method, which stops the training if three successive epochs return less than 0.001 improvement of the validation loss. Our default parameters were determined based on the trial and error using different parameters. The default values were set based on a small batch of different data sets, see Appendix A. The parameters are passed to the function `get_model` and can be changed by the user.

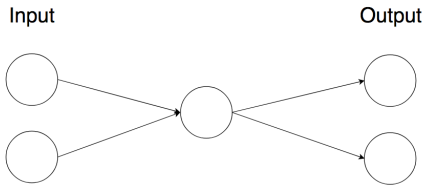


Figure 3.1: Autoencoder architecture at the first step.

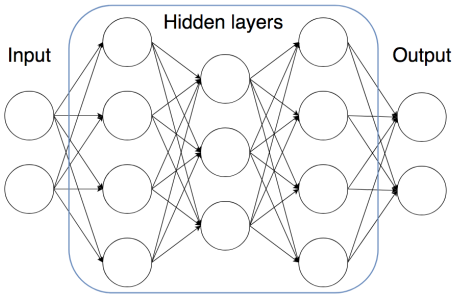


Figure 3.2: Autoencoder architecture at the second step.

Algorithm 1 Pseudocode of the dynamic autoencoder

$n \leftarrow$ input layer size (number of attributes)
$max\_layers \leftarrow$ maximum number of layers
$layer\_sizes \leftarrow$ empty list to save pre-calculated sizes of hidden layers
if n $<$ 100 **then**
    $layer\_sizes.append(2 * n)$
    $doubled \leftarrow$ **true**
end if//
for i = 0; i $<$ max_layers; i++ **do**
    if *doubled* **then**
        $layer\_sizes.append((2 * n) * r^{\mathrm{i}})$
    else
        $layer\_sizes.append(n * r^{\mathrm{i}}))$
    end if
end for// calculate hidden layer's size and add to list
for i = 0; i $<$ max_layers; i++ **do**
    $build\_encoder \leftarrow$ build encoder with i hidden layers, where j-th hidden layer has size of layer_sizes[j]
    $add\_new\_layer \leftarrow$ add middle layer with size layer_sizes[i]
    $build\_decoder \leftarrow$ build decoder with i hidden layers, where j-th hidden layer has size of layer_sizes[i-j]
    $train\_model$
    if current_loss + threshold $\geq$ previous_loss **then**
        $delete\_new\_layer \leftarrow$ delete the middle layer
        $delete\_first\_dec\_layer \leftarrow$ delete the first layer in decoder
        $retrain\_network \leftarrow$ retrain network without the deleted layers
        break
    end if
end for

3.2.2 Data Generating

The decoder of the trained autoencoder model is used to generate new data. Different inputs can be used to get generated data as an output. Firstly, samples from uniform distribution $\mathcal{U}(0, 1)$ and later from normal distribution $\mathcal{N}(0, 1)$ were used as an input. Both options were tested and results were not satisfying, therefore a better alternative was needed.

The better alternative is using both parts of the autoencoder – an encoder and a decoder. The encoder encodes the original data. The encoded data is used to get random samples from Sci-kit's function `kde`. `kde` is based on the kernel density estimation method [13], which estimates the probability density function of some distribution. The random samples are used as an input of the decoder. The decoder returns new generated data. The results of testing for choosing the input for generating data are in Appendix B.2.

3.3 Variational Autoencoders

Similarly to classical autoencoders, we want the solution to be as general as possible, therefore we developed dynamic variational autoencoders.

3.3.1 Model

The implementation of the dynamic variational autoencoder is very similar to the implementation of the classical dynamic autoencoders. The only difference in the structure is the middle hidden layer, where instead of one layer we have two layers. In the first layer, we have z_mean and z_log_var, together they present a latent space of the input data. The second part is a sampling layer, which samples similar points z from the latent normal distribution that is assumed to generate the data with $z = z\_mean + exp(z\_log\_var) * epsilon$, where epsilon is a random normal tensor. The Algorithm 2 shows the pseudo code of the dynamic variational autoencoder.

All the layers, except the output layer use the hyperbolical tangent activa-

Algorithm 2 Pseudocode of the dynamic variational autoencoder

$n \leftarrow$ input layer size (number of attributes)
$max\_layers \leftarrow$ maximum number of layers
$layer\_sizes \leftarrow$ empty list to save pre-calculated sizes of hidden layers
if n $<$ 100 **then**
    $layer\_sizes.append(2 * n)$
    $doubled \leftarrow$ **true**
end if//
for i = 0; i $<$ max_layers; i++ **do**
    if *doubled* **then**
        $layer\_sizes.append((2 * n) * r^{\mathrm{i}})$
    else
        $layer\_sizes.append(n * r^{\mathrm{i}}))$
    end if
end for// calculate hidden layer's size and add to list
for i = 0; i $<$ max_layers; i++ **do**
    $build\_encoder \leftarrow$ build encoder with i hidden layers, where j-th hidden layer has size of layer_sizes[j]
    $add\_latent\_layers \leftarrow$ add z_mean and z_log_var layers of size 2∗layer_sizes[i]
    $add\_sampling\_layer \leftarrow$ add sampling layer: $z\_mean + exp(z\_log\_var) * \mathcal{N}(0, 1)$ of size 2∗layer_sizes[i]
    $build\_decoder \leftarrow$ build decoder with i hidden layers, where j-th hidden layer has size of layer_sizes[i-j]
    $train\_model$
    if current_loss + threshold $\geq$ previous_loss **then**
        $delete\_new\_layers \leftarrow$ delete the middle layers
        $delete\_inner\_layers \leftarrow$ delete the last layer in encoder and the first layer in decoder
        $add\_latent\_sampling \leftarrow$ add latent and sampling layers of size 2*layer_sizes[i-1]
        $retrain\_network \leftarrow$ retrain network
        break
    end if
end for

tion function (`tanh`), while the output layer uses sigmoid activation function. The `r` parameter has a default value 0.1. Autoencoder is trained with the `rmsprop` algorithm [37] for a total of 500 epochs with the default batch size of 16. It uses a sum of binary cross-entropy and KL-divergence for a loss function, see Equation (2.4). The model uses early stopping method, which stops the training if five successive epochs return less than 0.001 improvement in the validation loss. As there were problems with overfitting, the dropout method was used. The default dropout rate is 0.2, which means we randomly drop 20% of the units in every layer, except in the input and the output layer. The default values were set based on a small batch of different data sets, see Appendix A. The parameters can be changed the users.

3.3.2 Data Generating

Generating data with variational autoencoders is easier in comparison to the autoencoders. To generate data we only need the second (decoding) part of the VAE, as the new data is decoded from the samples of the latent space. We tested three approaches – using samples from uniform distribution $\mathcal{U}(0, 1)$, from uniform distribution $\mathcal{U}(-1, 1)$ and from normal distribution $\mathcal{N}(0, 1)$. The last approach produced the best results. The results of testing for choosing the input for generating data are in Appendix B.2.

Chapter 4

Evaluation Scenarios

The generated data were tested and compared to the original data – both should have approximately the same structure, statistical properties (mean, standard deviation, skewness and kurtosis) and should return similar performance with applied machine learning methods.

The evaluation process starts with shuffling and splitting data in 10 subsets of equal size. We calculate the mean of the results through 10 iterations, where test set d_{test} in the i-th iteration is i-th subset and training set d_{train} is formed from all the other subset, see Figure 4.1. The training data is used to train a model for generating a new data d_{gen}. In every iteration, 1000 instances are generated. The whole evaluation process is repeated 5 times.

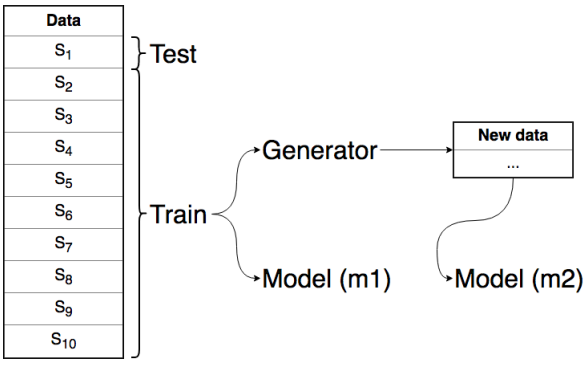


Figure 4.1: Graphic presentation of evaluation.

4.1 Data Sets

We used 52 data sets, which are listed in Table 4.1, where *no. of classes* presents a number of classes and column *attr. after encoding* presents a number of attributes after encoding categorical data. Most of the data sets have mixed attributes. The data sets with only numerical attributes were used to evaluate regression instead of classification performance. There are 17 data sets, which were used by Robnik-Šikonja [31] in order to compare the performance of both approaches.

The data sets have between 3 and 280 attributes with 16 to 20000 cases. Most data sets have around 500 cases, as we want to show, it is possible to generate data from scarce data.

4.2 Statistics of Attributes

Standard statistics of numerical attributes (mean, standard deviation, skewness, and kurtosis) were compared. The statistics were calculated on normalized data for each attribute separately. We report the mean of the difference across all attributes. The reported values are named m($\Delta mean$), m(Δstd), m($\Delta\gamma_1$) and m($\Delta\gamma_2$). In the ideal case, those values would be 0.

4.3 Clustering

Another method to determinate the similarity of two data sets is comparing the clusters in both data sets. KMeans algorithm, from the Sci-kit library, was used for the task.

The clustering is performed on training data d_{train} and generated data d_{gen}. We name the resulting clusters Cl_{train} and Cl_{gen}, respectively. In the next step, the closest cluster for each case in d_{test} is calculated based on both clusterings separately, which results in $Cl_{test|train}$ and $Cl_{test|gen}$. These two cluster assignments use the same instances and can be evaluated using Adjusted Rand index or shortly ARI. ARI has a value close to 0.0 for random

dataset	cases	attributes	numeric	nominal	no. of classes	attr. after encoding	source
aids	570	6	6	0	/	/	R datasets [3]
annealing	898	39	7	32	5	87	UCI [10]
balance-scale	625	5	4	1	3	7	UCI [10]
Benefits	4877	17	7	10	4	29	R datasets [3]
biomass	153	8	6	2	8	17	R datasets [3]
Bordeaux	72	3	3	0	/	/	Sheather [35]
breast-cancer	286	10	1	9	2	43	UCI [10]
breast-cancer-wdbc	569	31	30	1	2	32	UCI [10]
breast-cancer-wisconsin	699	10	8	2	2	21	UCI [10]
bridges-version1	108	12	1	11	8	160	UCI [10]
bridges-version2	108	12	0	12	8	102	UCI [10]
cars04	234	10	10	0	/	/	Sheather [35]
Caschool	420	15	13	2	45	60	R datasets [3]
Caterpillars	267	18	15	3	2	21	R datasets [3]
Crime	630	23	21	2	3	26	R datasets [3]
dermatology	366	35	33	2	6	100	UCI [10]
diamonds	2192	7	7	0	/	/	kaggle [2]
DoctorAUS	5190	15	13	2	4	20	R datasets [3]
ecoli	336	8	7	1	8	15	UCI [10]
Fatality	336	10	8	2	2	12	R datasets [3]
Fishing	1182	12	11	1	4	15	R datasets [3]
flags	194	29	25	4	8	56	UCI [10]
glass	214	10	9	1	6	15	UCI [10]
haberman	306	4	3	1	2	5	UCI [10]
highway	39	12	11	1	4	15	R datasets [3]
hla	271	8	7	1	2	9	R datasets [3]
honeyproduction	626	6	6	0	/	/	kaggle [23]
Hoops	147	21	20	1	9	29	R datasets [3]
house_data	21613	19	19	0	/	/	kaggle [1]
infant_mortality	105	4	2	2	4	8	R datasets [3]
InstInnovation	6208	24	22	2	2	26	R datasets [3]
insurance	1338	4	4	0	/	/	kaggle [7]
InsuranceVote	435	6	5	1	2	7	R datasets [3]
iris	150	5	4	1	3	7	UCI [10]
Kakadu	1827	22	16	6	3	29	R datasets [3]
longley	16	6	6	0	/	/	R datasets [3]
magazines	204	4	4	0	/	/	Sheather [35]
MedGPA	55	10	8	2	2	12	R datasets [3]
midwest	437	25	24	1	16	40	R datasets [3]
Mroz	753	18	16	2	2	20	R datasets [3]
msleep	83	8	6	2	5	16	R datasets [3]
pgatour2006	196	10	10	0	/	/	Sheather [35]
post-operative	90	9	0	9	3	27	UCI [10]
primary-tumor	339	18	12	6	21	50	UCI [10]
skulls	150	5	4	1	5	9	R datasets [3]
soils	48	14	11	3	3	30	R datasets [3]
soybean-large	307	36	1	35	19	150	UCI [10]
tic-tac-toe	958	10	0	10	2	29	UCI [10]
Tobacco	2724	9	7	2	3	13	R datasets [3]
weatherHistory	998	6	6	0	/	/	kaggle [5]
winequality-white	4898	10	10	0	/	/	UCI [10]

Table 4.1: Data sets used to evaluate the performance.

assigned clusters and 1.0 when the two clusterings are identical. A negative value means that the clusterings are less similar than what is expected from a random assignment to clusters.

4.4 Classification Performance

The idea of comparing data sets based on classification is to train models separately on the original and generated data. Approximately the same performance on both data sets of a model, trained on the original data, indicates that the generated data are within the original distribution. On the other side, approximately the same performance of a model, trained on the generated data, suggests that the generated data is a good substitute for original data regarding machine learning. Additionally, if a model, trained on the original data, shows a better performance on the generated data in comparison to the original data, we can conclude that the generator is oversimplified.

To classify the data, we used random forests. The Sci-kit library provides the `RandomForestClassifier` function. The classifier is trained on d_{train} and d_{gen}. The resulting models are named m_{train} and m_{gen}. The performance of the models is evaluated on the data, which was not seen during the training period (d_{test}). The performance is measured as accuracy of the model – a percentage of correctly classified cases. The reported values are **m1d1**, which is the performance of the models built on the original data and tested on the original data (m_{train} on d_{test}), and **m2d1**, which is the performance of the models built on the generated data and tested on the original data (m_{gen} on d_{test}). We also report $\Delta(m_1, m_2)$, which is calculated as $\Delta(m_1, m_2) = m2d1 - m1d1$.

4.5 Regression Performance

Comparing data sets based on regression is similar to comparing them based on classification. The goal is to have approximately the same performance

of both models, a model built on the original data and a model built on the generated data.

As a regression model we used random forests. The Sci-kit library provides the `RandomForestRegressor` function. The classifier is trained on d_{train} and d_{gen}. The performance of the models is evaluated on the data, which was not seen during the training period (d_{test}). The performance is measured as R^2 score, which is a statistical measure of how close the data are to the fitted regression line. A perfect model that always predicts the expected value, would get a R^2 score of 1.0. Same as before, the reported values are for **m1d1** and **m2d1**.

4.6 Testing Environment

The testing was done on Google Cloud Platform. We used a virtual machine with Ubuntu 16.04 operating system. The script ran on a single core of Intel Xeon CPU with 2.50 GHz. The machine had 16 GB RAM. The solution was developed and executed using Python 3.6.

Chapter 5

Results

The results are presented in tabular form. The tables show the metrics explained in the previous section. We also report the time in seconds used for generating data for one training set ($t[s]$), and the percentage of generated data exactly equal to cases from the train set (=).

The results are split in three parts. In the first part, generators' performances were tested on data with mixed attributes, which can be seen in Table 5.1 and Table 5.2. We wanted to test our solution on data sets with different number of attributes and instances to get a better overview of generator performance. On average, a generator based on autoencoders needs 13.1 seconds to generate data, while variational autoencoders needs 17.2 seconds. Overall, the performance is better for autoencoders with average ARI value of 0.73 and average difference between $m1d1$ and $m2d1$ is -10%. With variational autoencoders, the ARI value is 0.43 and $\Delta(m_1, m_2)$ is -23%. VAEs give significantly worse results for 22 out of 23 data sets. Wilcoxon signed-rank test at $\alpha = 0.05$ shows that the median difference in $\Delta(m1, m2)$ between generators is not zero and supports the alternative hypothesis that autoencoders perform better than variational autoencoders. For autoencoders, both models, trained on the original and generated data, perform almost equally well on three data sets. Additionally, the difference in accuracy is less than 5 percentage points in 6 cases. These data sets, on which autoencoders per-

form well, do not seem to have any distinctive characteristic different from other data sets, on which the performance is worse.

In the second part, we tested generators on 17 data sets used by Robnik-Šikonja [31]. The aim was to compare results to generators based on RBF networks and random forests. We decided to compare only autoencoders, as they produce better results, as seen in the previous test. Table 5.3 shows results of autoencoders based generators and Table 5.4 of RBF and random forests based generators. Wilcoxon signed-rank test at $\alpha = 0.05$ shows that the median difference in $\Delta(m1, m2)$ between generators is not zero and supports the alternative hypothesis that generators based on RBF networks and random forests perform better. Our generator provides slightly better results for 2 data sets and similar results for 7 data sets. We got slightly worse results for 3 datasets and much worse results for 5.

To get an idea when the generators produce acceptable data, we analyzed how are data sets characteristics correlated to the difference between $m1d1$ and $m2d1$ ($\Delta(m_1, m_2)$). Data sets characteristics used are type of attributes (numerical, categorical or mixed), number of attributes, number of cases, number of class values and normalized Shannon entropy of the class variable. The Shannon entropy tells us the information encoded in the distribution of class values (value 1 means that the ratios of all classes are equal). It is calculated as:

$$E = \frac{\sum_{i=1}^{N} p_i \ log_2 \ (\frac{1}{p_i})}{log_2 \ (N)}, \text{ where } p_i = \text{ probability of class } i$$

Figure 5.1.a presents the correlation matrix for the generators based on autoencoders. It can be seen there is no strong correlation between characteristics and the difference in performance. The matrix suggests that increasing the number of classes increases the margin between models' performance and increasing the number of cases decreases the margin. Type of the attributes also has impact on the performance. From the matrix it seems that the generators perform better for categorical data than numerical data. Nonetheless, they work best on the mixed data, since attribute types are integers, where

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
Benefits	27.5	0	0.023	0.055	0.20	1.01	0.56 ± 0.04	50.52 ± 2.40	46.56 ± 2.60	−3.95
biomass	9.7	0	0.018	0.021	0.98	8.31	0.94 ± 0.03	78.02 ± 10.21	63.09 ± 11.00	−14.93
Caschool	16.0	0	0.022	0.035	0.23	1.63	0.52 ± 0.06	21.24 ± 6.60	14.52 ± 4.55	−6.71
Caterpillars	6.3	0	0.012	0.017	0.17	0.28	0.85 ± 0.08	84.65 ± 7.84	69.06 ± 12.52	−15.59
Crime	10.1	0	0.022	0.062	0.76	4.72	0.89 ± 0.03	89.11 ± 4.46	63.17 ± 7.68	−25.94
DoctorAUS	40.2	0	0.031	0.060	0.42	3.87	0.87 ± 0.03	55.36 ± 1.92	42.57 ± 5.99	−12.80
Fatality	9.0	0	0.024	0.056	0.19	1.26	0.62 ± 0.05	94.15 ± 3.75	81.12 ± 6.81	−13.04
Fishing	13.2	0	0.046	0.058	0.55	2.54	0.96 ± 0.03	91.79 ± 2.66	63.43 ± 5.48	−28.36
highway	5.1	0	0.023	0.036	0.40	1.62	0.56 ± 0.09	61.83 ± 22.37	69.33 ± 23.83	7.50
hla	8.8	0	0.017	0.022	0.10	0.16	0.65 ± 0.06	99.85 ± 1.04	99.93 ± 0.52	0.07
Hoops	6.8	0	0.016	0.050	0.11	0.93	0.42 ± 0.05	22.46 ± 10.33	12.37 ± 7.33	−10.09
infant_mortality	10.1	0	0.020	0.032	1.01	6.86	0.94 ± 0.07	56.75 ± 15.57	43.89 ± 14.03	−12.85
InstInnovation	31.1	0	0.017	0.058	1.37	18.28	0.75 ± 0.04	95.28 ± 0.83	86.87 ± 2.29	−8.41
InsuranceVote	9.3	0	0.025	0.057	0.14	1.38	0.69 ± 0.10	87.95 ± 5.27	89.05 ± 7.21	1.10
iris	7.0	0	0.021	0.017	0.19	0.33	0.56 ± 0.07	94.93 ± 4.73	81.73 ± 9.69	−13.20
Kakadu	19.5	0	0.026	0.032	0.19	0.56	0.53 ± 0.04	99.69 ± 0.43	99.50 ± 0.67	−0.20
MedGPA	5.3	0	0.020	0.031	0.25	1.38	0.63 ± 0.07	75.73 ± 15.89	70.60 ± 18.82	−5.13
midwest	9.8	0	0.013	0.025	1.37	12.31	0.93 ± 0.04	90.62 ± 4.25	62.48 ± 7.29	−28.14
Mroz	13.0	0	0.025	0.078	0.28	1.10	0.72 ± 0.05	69.77 ± 4.70	57.98 ± 6.83	−11.80
msleep	6.6	0	0.013	0.034	1.17	7.06	0.80 ± 0.08	54.81 ± 17.40	40.83 ± 17.72	−13.97
skulls	7.1	0	0.019	0.035	0.13	0.86	0.74 ± 0.04	24.27 ± 11.29	24.93 ± 10.65	0.67
soils	6.3	0	0.016	0.010	0.12	0.34	0.68 ± 0.07	94.50 ± 11.63	98.80 ± 4.75	4.30
Tobacco	23.1	0	0.029	0.065	0.23	1.85	0.99 ± 0.01	66.81 ± 2.82	58.50 ± 4.27	−8.30
avg	13.1	0	0.022	0.041	0.46	3.42	0.73 ± 0.17	72.18 ± 26.24	62.62 ± 26.67	−9.55

Table 5.1: Autoencoder results on data sets with mixed types of attributes.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
Benefits	53.3	0	0.009	0.140	0.34	0.73	0.36 ± 0.04	50.09 ± 2.10	42.30 ± 2.77	−7.79
biomass	9.0	0	0.290	0.053	4.01	23.29	0.52 ± 0.15	78.89 ± 10.61	47.65 ± 16.63	−31.24
Caschool	10.7	0	0.135	0.109	1.02	1.80	0.00 ± 0.00	22.29 ± 6.09	5.71 ± 3.87	−16.57
Caterpillars	7.1	0	0.100	0.154	0.50	0.90	0.56 ± 0.12	84.78 ± 8.30	54.13 ± 10.74	−30.65
Crime	11.2	0	0.225	0.076	2.23	7.25	0.56 ± 0.06	88.29 ± 4.70	45.71 ± 8.31	−42.57
DoctorAUS	60.3	0	0.032	0.107	1.16	4.25	0.72 ± 0.06	55.36 ± 2.17	40.93 ± 4.39	−14.44
Fatality	8.2	0	0.111	0.107	0.66	0.93	0.38 ± 0.09	92.96 ± 3.63	63.69 ± 10.90	−29.27
Fishing	16.9	0	0.113	0.119	1.30	4.72	0.59 ± 0.05	92.08 ± 3.31	45.06 ± 6.04	−47.02
highway	6.1	0	0.210	0.114	1.34	2.59	0.27 ± 0.26	58.00 ± 25.16	32.50 ± 24.17	−25.50
hla	7.4	0	0.066	0.198	0.39	0.70	0.44 ± 0.11	100.00 ± 0.00	91.43 ± 7.80	−8.57
Hoops	5.8	0	0.067	0.101	0.29	0.49	0.10 ± 0.08	17.68 ± 8.75	14.76 ± 10.26	−2.91
infant_mortality	7.7	0	0.276	0.068	2.14	8.84	0.38 ± 0.20	54.36 ± 15.97	29.60 ± 16.76	−24.76
InstInnovation	62.7	0	0.014	0.063	2.69	26.37	0.65 ± 0.04	95.22 ± 0.83	76.43 ± 6.04	−18.79
InsuranceVote	9.1	0	0.081	0.111	0.63	1.18	0.59 ± 0.06	87.55 ± 5.09	84.22 ± 9.53	−3.33
iris	8.1	0	0.020	0.152	0.26	0.64	0.47 ± 0.07	94.93 ± 5.09	74.93 ± 14.27	−20.00
Kakadu	29.0	0	0.067	0.141	0.42	0.59	0.30 ± 0.04	99.79 ± 0.34	77.94 ± 5.17	−21.85
MedGPA	5.4	0	0.070	0.086	0.50	1.26	0.46 ± 0.25	75.47 ± 16.12	58.20 ± 22.81	−17.27
midwest	9.6	0	0.380	0.081	3.90	21.78	0.05 ± 0.09	89.89 ± 4.03	34.42 ± 20.95	−55.47
Mroz	12.7	0	0.130	0.099	0.49	0.89	0.58 ± 0.03	69.38 ± 6.15	51.92 ± 5.77	−17.46
msleep	6.5	0	0.238	0.064	2.46	9.55	0.12 ± 0.14	57.14 ± 15.58	37.64 ± 16.48	−19.50
skulls	7.9	0	0.019	0.110	0.10	0.35	0.60 ± 0.16	24.13 ± 10.99	21.73 ± 11.30	−2.40
soils	6.2	0	0.138	0.128	0.36	0.77	0.13 ± 0.18	92.70 ± 13.83	31.10 ± 21.89	−61.60
Tobacco	34.1	0	0.059	0.074	0.66	2.95	0.98 ± 0.02	66.55 ± 3.20	52.78 ± 7.62	−13.77
avg	17.2	0	0.124	0.107	1.21	5.34	0.43 ± 0.23	71.63 ± 24.82	48.47 ± 21.75	−23.16

Table 5.2: Variational autoencoders results on data sets with mixed types of attributes.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
annealing	18.1	0	0.013	0.021	0.27	1.19	0.67 ± 0.04	99.31 ± 1.06	89.12 ± 15.62	-10.19
balance-scale	11.4	0	0.016	0.023	0.06	0.32	0.66 ± 0.08	81.09 ± 4.71	77.70 ± 4.69	-3.39
breast-cancer	8.9	0	0.022	0.024	0.09	0.51	0.68 ± 0.04	70.26 ± 7.34	63.34 ± 10.47	-6.91
breast-cancer-wdbc	9.2	0	0.024	0.070	0.38	2.48	0.44 ± 0.04	94.98 ± 2.07	91.35 ± 3.84	-3.62
breast-cancer-wisconsin	15.0	0	0.031	0.023	0.23	0.80	0.78 ± 0.07	96.05 ± 2.30	93.70 ± 2.91	-2.35
bridges-version1	5.4	0	0.017	0.014	0.11	0.66	0.53 ± 0.04	64.75 ± 11.76	61.22 ± 10.73	-3.53
bridges-version2	6.0	36	0.000	0.000	0.00	0.00	0.57 ± 0.04	63.91 ± 13.34	59.11 ± 15.11	-4.80
dermatology	10.8	0	0.015	0.016	0.22	0.67	0.82 ± 0.03	96.23 ± 2.97	89.13 ± 6.46	-7.10
ecoli	7.9	0	0.014	0.044	0.29	1.00	0.71 ± 0.04	84.23 ± 5.81	53.95 ± 14.26	-30.28
flags	7.2	0	0.011	0.025	0.15	0.52	0.60 ± 0.04	60.81 ± 11.64	36.29 ± 12.07	-24.52
glass	7.5	0	0.025	0.062	0.70	3.68	0.85 ± 0.06	73.82 ± 8.69	40.69 ± 13.02	-33.13
haberman	9.0	0	0.021	0.052	0.18	0.57	0.53 ± 0.06	68.18 ± 6.82	68.69 ± 9.20	0.51
iris	6.8	0	0.021	0.013	0.19	0.31	0.52 ± 0.06	95.33 ± 6.00	83.07 ± 11.25	-12.27
post-operative	5.7	98	0.000	0.000	0.00	0.00	0.33 ± 0.03	61.56 ± 14.44	46.89 ± 17.40	-14.67
primary-tumor	10.0	0	0.011	0.025	0.09	0.24	0.66 ± 0.04	42.61 ± 8.19	23.56 ± 9.28	-19.05
soybean-large	11.3	20	0.011	0.025	0.16	0.85	0.70 ± 0.04	88.23 ± 13.45	20.36 ± 31.17	-67.87
tic-tac-toe	17.8	95	0.000	0.000	0.00	0.00	0.54 ± 0.04	95.95 ± 2.18	53.88 ± 5.11	-42.07
avg	9.9	15.3	0.015	0.026	0.18	0.81	0.62 ± 0.13	78.66 ± 16.33	61.89 ± 22.71	-16.78

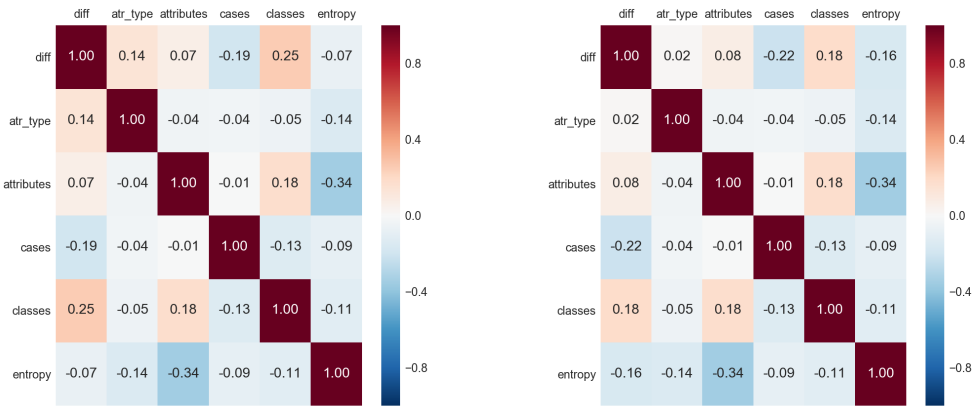
Table 5.3: Results of autoencoders (data sets from [31]).

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
annealing	/	0	0.126	0.014	0.65	1.29	0.63 ± 0.06	99.33 ± 1.00	95.55 ± 2.67	-3.79
balance-scale	/	63	0.018	0.043	0.07	0.26	0.53 ± 0.07	81.41 ± 3.54	71.27 ± 5.52	-10.14
breast-cancer	/	52	0.009	0.015	0.03	0.15	0.68 ± 0.08	70.87 ± 6.90	72.10 ± 7.78	1.23
breast-cancer-wdbc	/	0	0.017	0.033	0.24	1.10	0.41 ± 0.07	95.99 ± 2.13	93.28 ± 3.05	-2.70
breast-cancer-wisconsin	/	14	0.025	0.031	0.18	0.32	0.93 ± 0.02	95.83 ± 2.20	95.20 ± 2.25	-0.63
bridges-version1	/	0	0.008	0.025	0.15	0.48	0.52 ± 0.19	65.41 ± 13.83	63.57 ± 13.06	-1.84
bridges-version2	/	36	0.000	0.000	0.00	0.00	0.48 ± 0.13	65.74 ± 12.13	62.30 ± 11.05	-3.44
dermatology	/	0	0.053	0.031	0.30	0.68	0.68 ± 0.07	96.62 ± 3.15	93.93 ± 3.84	-2.69
ecoli	/	0	0.047	0.021	0.27	0.58	0.91 ± 0.06	84.82 ± 5.48	73.62 ± 9.30	-11.20
flags	/	0	0.015	0.016	0.17	0.57	0.67 ± 0.10	62.44 ± 9.68	47.76 ± 11.77	-14.68
glass	/	0	0.028	0.023	0.47	2.29	0.50 ± 0.17	76.17 ± 9.02	46.34 ± 10.18	-29.83
haberman	/	3	0.042	0.025	0.22	0.28	0.53 ± 0.12	67.54 ± 7.48	66.70 ± 8.41	-0.84
iris	/	0	0.034	0.020	0.17	0.21	0.54 ± 0.12	94.93 ± 5.26	91.20 ± 7.23	-3.73
post-operative	/	91	0.000	0.000	0.00	0.00	0.19 ± 0.11	61.84 ± 12.80	59.54 ± 12.54	-2.30
primary-tumor	/	43	0.014	0.008	0.10	0.22	0.48 ± 0.06	32.23 ± 5.74	31.61 ± 7.02	-0.62
soybean-large	/	14	0.019	0.022	0.30	1.45	0.67 ± 0.06	76.86 ± 18.10	76.63 ± 16.74	-0.23
tic-tac-toe	/	77	0.000	0.000	0.00	0.00	0.55 ± 0.04	95.26 ± 2.47	93.38 ± 2.83	-1.88
avg	/	2	0.027	0.019	0.20	0.58	0.58 ± 0.17	77.84 ± 17.42	72.59 ± 18.94	-5.25

Table 5.4: Results of generators based on RBF and random forests (data sets from [31]).

−1 stands for numerical data, 1 for categorical data and 0 for mixed data. Number of attributes and the entropy have a small impact on the difference in performance.

Figure 5.1.b presents the correlation matrix for the generators based on variational autoencoders. Same as before, it can be seen there is no strong correlation between characteristics and the difference between performance on original and generated data. The matrix suggests that increasing the number of cases and higher entropy decrease the margin between models' performance, while increasing number of classes increases the margin. Like before, generators work best with mixed data. We can conclude that a generator based on variational autoencoders would perform best on a data set with small number of mixed attributes with only two balanced classes.



a) b)

Figure 5.1: Correlation matrix of characteristics of data sets and $\Delta(m_1, m_2)$, reported as diff, for a) autoencoders and b) variational autoencoders.

Testing Regression Data Sets

In the last part, we tested performance on data sets with only numerical attributes and numerical predictor. The results are in Tables 5.5 and 5.6. The goal was to assess how good the generators handle regression problems. The results show that our generators could be a good approach, for some regression problems. Even though the average $\Delta(m1, m2)$ value of generators suggest that autoencoders perform better, comparing individual data set suggests VAEs are better. The average is misleading, as VAEs give significantly worse results on data sets, where both generators perform badly. The results show there is a high variance in generated data, therefore it is difficult to say which approach is better. Conducting the Wilcoxon signed-rank test we fail to reject the null hypothesis at $\alpha = 0.05$ that the median difference in $\Delta(m1, m2)$ between generators is zero. The generated data was acceptable for 3 data sets out of 12. In 6 cases models built on the generated data produce negative R^2, which means that the chosen model does not follow the trend of the data and fits the data worse than a horizontal line, representing the mean.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
aids	9.4	0	0.017	0.036	0.08	0.33	0.74 ± 0.08	0.89 ± 0.12	-1.06 ± 1.81	-1.95
Bordeaux	6.3	0	0.094	0.070	0.44	0.81	0.37 ± 0.11	-1.99 ± 7.33	-84.59 ± 263.94	-82.60
cars04	5.1	0	0.031	0.061	0.23	1.41	0.45 ± 0.07	0.82 ± 0.11	0.32 ± 0.46	-0.50
diamonds	14.4	0	0.032	0.139	1.20	3.19	0.32 ± 0.05	0.98 ± 0.01	0.60 ± 0.26	-0.38
honeyproduction	10.7	0	0.061	0.080	1.19	6.60	0.43 ± 0.08	0.75 ± 0.14	0.62 ± 0.22	-0.14
house_data	77.5	0	0.037	0.116	0.40	2.23	0.41 ± 0.07	0.87 ± 0.02	-0.24 ± 0.73	-1.11
insurance	15.1	0	0.047	0.107	0.14	1.07	0.41 ± 0.04	-0.15 ± 0.11	-0.99 ± 0.60	-0.84
longley	4.1	0	0.071	0.180	0.28	0.74	0.30 ± 0.10	-0.27 ± 3.53	-19.15 ± 83.85	-18.88
magazines	9.4	0	0.071	0.068	2.73	19.95	0.58 ± 0.12	0.66 ± 0.24	0.04 ± 1.51	-0.62
pgatour2006	6.6	0	0.024	0.090	0.16	1.32	0.27 ± 0.04	0.07 ± 0.51	-1.39 ± 2.86	-1.46
weatherHistory	12.2	0	0.025	0.086	0.17	1.16	0.39 ± 0.05	1.00 ± 0.00	0.82 ± 0.17	-0.18
winequality-white	28.2	0	0.050	0.158	0.38	3.73	0.26 ± 0.03	0.90 ± 0.01	0.12 ± 0.27	-0.78
avg	16.6	0	0.047	0.099	0.62	3.54	0.41 ± 0.13	0.38 ± 0.83	-8.74 ± 23.47	-9.12

Table 5.5: Results of autoencoders for regression problems.

We can conclude that our approach strongly depends on the data set for regression problems. We suspect that the results would improve if we used the mean squared error as a loss function in autoencoder and variational

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
aids	10.4	0	0.078	0.173	0.22	0.74	0.44 ± 0.07	0.89 ± 0.13	-3.37 ± 3.51	-4.26
Bordeaux	5.5	0	0.040	0.110	0.36	0.58	0.29 ± 0.23	-2.63 ± 8.56	-151.86 ± 355.80	-149.24
cars04	5.7	0	0.100	0.085	0.50	0.78	0.40 ± 0.12	0.82 ± 0.11	-0.82 ± 1.38	-1.65
diamonds	15.0	0	0.096	0.093	1.41	2.34	0.17 ± 0.02	0.98 ± 0.01	0.91 ± 0.03	-0.07
honeyproduction	9.0	0	0.234	0.039	2.38	8.65	0.14 ± 0.04	0.74 ± 0.15	0.63 ± 0.12	-0.10
house_data	160.5	0	0.006	0.081	0.92	1.74	0.53 ± 0.05	0.87 ± 0.02	0.34 ± 0.04	-0.53
insurance	16.6	0	0.025	0.110	0.20	0.45	0.55 ± 0.07	-0.18 ± 0.10	-0.02 ± 0.11	0.16
longley	5.2	0	0.039	0.197	0.21	0.42	0.95 ± 0.15	-0.79 ± 7.33	-18.53 ± 64.49	-17.74
magazines	9.1	0	0.212	0.066	3.41	21.87	0.18 ± 0.11	0.61 ± 0.32	-0.07 ± 0.92	-0.68
pgatour2006	5.0	0	0.025	0.089	0.13	0.45	0.14 ± 0.08	0.05 ± 0.71	-40.25 ± 30.79	-40.29
weatherHistory	13.8	0	0.058	0.112	0.34	0.46	0.32 ± 0.04	1.00 ± 0.00	0.90 ± 0.02	-0.10
winequality-white	29.1	0	0.003	0.070	0.77	3.99	0.35 ± 0.03	0.90 ± 0.01	0.59 ± 0.03	-0.31
avg	23.7	0	0.076	0.102	0.90	3.54	0.37 ± 0.22	0.27 ± 1.02	-17.63 ± 42.14	-17.90

Table 5.6: Results of variational autoencoders based for regression problems.

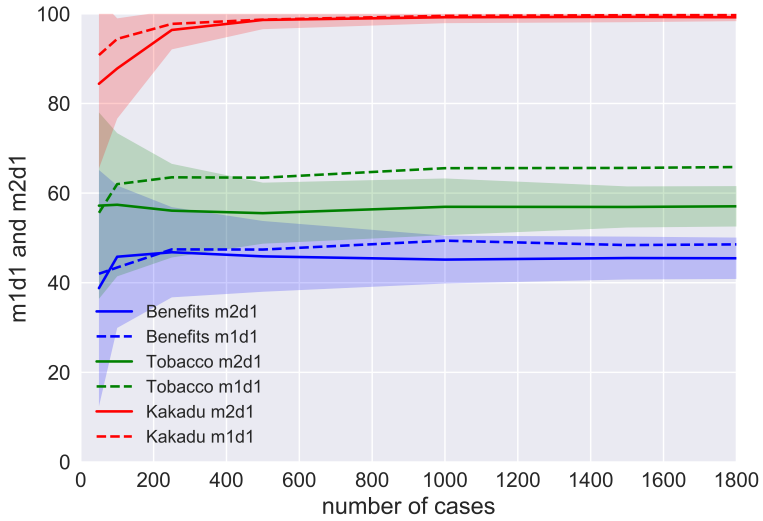
autoencoders, as mean squared error is usually used with regression problems. Since our focus is more on classification tasks, improvements of generators for regression problems is left for future work.

5.1 Dependence on the Number of Cases

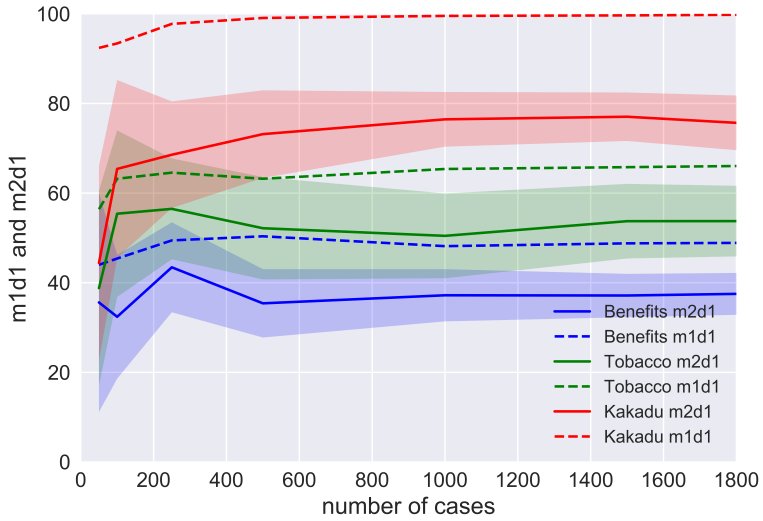
We tested the hypothesis that increasing the number of training instances significantly improves the results. We selected three data sets with at least 1500 instances (Benefits, Kakadu and Tobacco). The data sets were tested with 50, 100, 250, 500, 1000 and 1500 training instances, which were randomly selected from the data set. Figure 5.2.a shows m1d1 values (dashed line) and m2d1 values (solid line) of autoencoders and their standard deviations for each data set. It can be seen that average values stop improving with approximately 300 cases and the standard deviations stabilize after 1000 cases. On the other hand, generators based on VAEs (Figure 5.2.b) seem to be more volatile, especially when there are less than 500 cases.

We can conclude that the generators' performance is unstable with high variance for small number of instances. The performance improves and stabilizes after reaching a certain threshold. We assume that the threshold is around 1000 instances, although we cannot be sure, as we only tested on

three different data sets.



a)



b)

Figure 5.2: Graph of m2d1 classification accuracy with its standard deviation depending on the number of cases for a) autoencoders and b) variational autoencoders.

5.2 Learning Parameters

We assume that the performance of generators could be improved by implementing a system to set parameters specifically for each data set. A random forest prediction model was built for each parameter (epoch, batch size, `r` parameter and drop rate). Training data were results obtained during testing for setting the default parameters, see Appendix A. The predictions for each parameter are based on the number of cases, the number of attributes, the number of classes and the class entropy of data sets. The possible values of each parameter are in Table 5.7. We expect to get better results in comparison to results with default parameters, even though there are only 13 cases in the training data and we predict each parameter separately assuming the parameters are independent from each other.

parameter	possible values
batch size	16, 32, 64, 128, 256
drop rate	0.0, 0.1, 0.2, 0.3, 0.4, 0.5
epochs	5, 10, 20, 50, 100
`r`	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

Table 5.7: Possible values of the parameters.

Tables 5.8 and 5.9 show the results of testing on 23 data sets with mixed attributes. Parameter prediction for autoencoders produces better results for 7 data sets out of 23. With variational autoencoders we get better results for 6 data sets. Wilcoxon signed-rank test at $\alpha = 0.05$ shows that the median difference is not zero for $\Delta(m1, m2)$ using autoencoder generators with default and predicted parameters. This supports the alternative hypothesis that autoencoders with default parameters give better results. We get the same conclusion for VAEs.

The results were surprising as we expected better results with predicted parameters. On the other hands, the results make sense, because the model was built on only 13 datasets. We expect that the results would improve if

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
Benefits	37.3	0	0.010	0.160	0.85	1.31	0.49 ± 0.06	49.85 ± 2.22	47.97 ± 3.39	-1.89
biomass	10.3	0	0.018	0.084	1.28	9.15	0.89 ± 0.10	77.03 ± 11.47	60.91 ± 10.57	-16.13
Caschool	5.6	0	0.013	0.021	0.34	1.53	0.38 ± 0.04	22.81 ± 5.64	11.62 ± 5.21	-11.19
Caterpillars	7.4	0	0.020	0.048	0.19	0.17	0.65 ± 0.09	86.06 ± 5.99	80.96 ± 6.84	-5.10
Crime	11.6	0	0.020	0.062	1.55	5.33	0.68 ± 0.10	88.63 ± 4.94	60.29 ± 6.96	-28.35
DoctorAUS	36.0	0	0.011	0.124	1.95	6.50	0.76 ± 0.05	55.38 ± 1.99	51.17 ± 3.83	-4.20
Fatality	6.0	0	0.021	0.096	0.63	1.05	0.51 ± 0.11	93.56 ± 4.25	74.90 ± 8.64	-18.66
Fishing	17.8	0	0.024	0.054	1.22	2.50	0.92 ± 0.05	91.96 ± 2.54	61.69 ± 5.18	-30.27
highway	4.6	0	0.031	0.116	0.79	1.98	0.46 ± 0.11	59.50 ± 21.80	48.50 ± 26.86	-11.00
hla	10.8	0	0.021	0.210	0.37	0.94	0.56 ± 0.07	99.93 ± 0.52	96.54 ± 6.96	-3.39
Hoops	3.4	0	0.060	0.083	0.28	0.49	0.10 ± 0.04	18.99 ± 10.34	12.11 ± 8.47	-6.88
infant_mortality	6.6	0	0.047	0.084	1.39	7.49	0.73 ± 0.23	56.93 ± 12.80	32.53 ± 14.81	-24.40
InstInnovation	9.4	0	0.008	0.057	2.94	24.61	0.62 ± 0.02	95.29 ± 0.77	77.99 ± 3.80	-17.29
InsuranceVote	12.9	0	0.021	0.149	1.26	1.38	0.60 ± 0.04	87.04 ± 5.02	86.19 ± 7.17	-0.84
iris	5.7	0	0.054	0.107	0.34	0.58	0.47 ± 0.05	94.80 ± 5.54	59.60 ± 16.99	-35.20
Kakadu	5.1	0	0.019	0.149	0.37	0.60	0.42 ± 0.05	99.73 ± 0.45	77.84 ± 3.72	-21.88
MedGPA	4.2	0	0.056	0.128	0.67	1.12	0.43 ± 0.14	77.27 ± 17.03	50.73 ± 22.37	-26.53
midwest	10.0	0	0.009	0.029	1.57	12.52	0.97 ± 0.01	89.75 ± 5.19	72.68 ± 6.54	-17.06
Mroz	12.3	0	0.019	0.078	0.28	0.85	0.86 ± 0.03	68.28 ± 4.62	62.76 ± 6.39	-5.52
msleep	5.8	0	0.014	0.039	1.41	7.62	0.76 ± 0.11	58.36 ± 20.15	36.94 ± 17.62	-21.42
skulls	5.4	0	0.020	0.106	0.30	0.33	0.59 ± 0.12	22.40 ± 10.98	22.00 ± 12.45	-0.40
soils	4.7	0	0.018	0.186	0.54	1.07	0.32 ± 0.07	92.90 ± 11.45	37.00 ± 21.59	-55.90
Tobacco	7.5	0	0.012	0.085	0.94	3.26	0.87 ± 0.08	66.71 ± 2.30	63.94 ± 3.30	-2.77
avg	10.4	0	0.024	0.098	0.93	4.02	0.61 ± 0.21	71.88 ± 24.71	55.95 ± 22.33	-15.92

Table 5.8: Autoencoder results on data sets with mixed types of attributes with parameters prediction.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
Benefits	5.7	0	0.084	0.125	0.53	0.88	0.18 ± 0.04	50.28 ± 2.04	35.60 ± 4.80	-14.67
biomass	7.0	0	0.397	0.027	4.46	23.11	0.17 ± 0.17	77.97 ± 12.07	15.62 ± 15.95	-62.35
Caschool	6.7	0	0.155	0.098	1.06	1.84	0.00 ± 0.00	21.81 ± 5.33	5.10 ± 3.98	-16.71
Caterpillars	6.6	0	0.096	0.114	0.51	0.83	0.39 ± 0.11	85.91 ± 5.55	63.77 ± 10.90	-22.13
Crime	5.1	0	0.306	0.071	2.39	7.53	0.06 ± 0.04	89.40 ± 3.89	37.43 ± 8.28	-51.97
DoctorAUS	8.7	0	0.327	0.137	2.14	7.02	0.26 ± 0.06	55.65 ± 1.77	47.28 ± 3.22	-8.37
Fatality	5.6	0	0.163	0.062	0.73	1.10	0.31 ± 0.07	93.75 ± 4.60	55.91 ± 12.76	-37.84
Fishing	6.0	0	0.310	0.093	1.77	5.27	0.12 ± 0.08	91.66 ± 2.66	29.97 ± 9.46	-61.68
highway	5.2	0	0.222	0.106	1.37	2.55	0.27 ± 0.26	62.50 ± 21.29	26.83 ± 21.30	-35.67
hla	6.0	0	0.101	0.204	0.50	0.75	0.36 ± 0.09	100.00 ± 0.00	69.19 ± 16.45	-30.81
Hoops	4.7	0	0.061	0.061	0.30	0.69	0.06 ± 0.06	20.04 ± 10.35	12.41 ± 8.96	-7.63
infant_mortality	5.8	0	0.343	0.073	2.38	8.96	0.52 ± 0.22	55.69 ± 12.35	27.22 ± 17.73	-28.47
InstInnovation	10.2	0	0.292	0.109	3.10	28.04	0.45 ± 0.11	95.17 ± 0.72	71.78 ± 7.26	-23.39
InsuranceVote	5.6	0	0.126	0.081	0.67	1.45	0.53 ± 0.07	87.46 ± 5.46	67.04 ± 15.29	-20.42
iris	7.6	0	0.027	0.134	0.27	0.57	0.49 ± 0.07	94.53 ± 5.76	72.80 ± 15.65	-21.73
Kakadu	8.6	0	0.133	0.195	0.55	0.83	0.29 ± 0.04	99.82 ± 0.28	73.00 ± 7.01	-26.82
MedGPA	5.0	0	0.073	0.078	0.52	1.29	0.34 ± 0.27	78.20 ± 18.19	49.00 ± 20.19	-29.20
midwest	5.0	0	0.405	0.081	3.94	22.17	0.00 ± 0.01	90.39 ± 4.17	42.70 ± 15.64	-47.69
Mroz	5.2	0	0.188	0.086	0.64	1.00	0.48 ± 0.08	68.66 ± 4.52	49.16 ± 7.81	-19.49
msleep	6.1	0	0.270	0.051	2.57	9.53	0.24 ± 0.16	57.94 ± 17.57	28.58 ± 19.50	-29.36
skulls	9.6	0	0.015	0.123	0.09	0.35	0.73 ± 0.23	22.40 ± 11.06	23.73 ± 8.96	1.33
soils	4.9	0	0.144	0.127	0.38	0.75	0.16 ± 0.23	90.90 ± 13.66	30.20 ± 24.33	-60.70
Tobacco	12.1	0	0.228	0.080	1.27	3.28	0.82 ± 0.06	66.68 ± 2.58	55.87 ± 6.38	-10.82
avg	6.7	0	0.194	0.101	1.40	5.64	0.32 ± 0.21	72.03 ± 24.65	43.05 ± 20.18	-28.98

Table 5.9: Variational autoencoders results on data sets with mixed types of attributes with parameters prediction.

more data sets and more values of parameters were available.

5.2.1 Grid Search

Predicting parameters independently from each other produced worse results than using the default parameters for each data set. The parameters strongly depend on each other and the data set, therefore we implemented grid search. We tested 10 data sets with 216 combinations of the parameters (Table 5.10). We selected 10 data sets from Table 5.1 based on the average execution time to make testing as quick as possible. Since training neural networks requires a lot of computational power, we tested the data sets using 3-fold cross validation instead 5 x 10-fold cross validation.

parameter	values
activation	tanh, relu
batch size	16, 64, 256
drop rate	0.0, 0.2, 0.4
epochs	20, 50, 100
`r`	0.10, 0.25, 0.50, 0.75

Table 5.10: Possible values of the parameters.

Tables 5.11 show the best result for each data set using gird search for generators based on autoencoders. The best result was selected based on the $\Delta(m_1, m_2)$ score. The $\Delta(m_1, m_2)$ for 5 data sets is almost zero, which means the generated data is almost equivalent to the original data. Wilcoxon signed-rank test at $\alpha = 0.05$ shows that the median difference in $\Delta(m1, m2)$ between results with default parameters and grid search is not zero and supports the alternative hypothesis that grid search returns significantly better results.

We showed that the grid search significantly improves the results. We decided to build a model for predicting efficient parameters. The model was built on the results obtained during the grid search. A random forest from Sci-kit learn library was used to predict the $\Delta(m1, m2)$ based on the number

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
biomass	20.0	0	0.010	0.032	1.43	10.39	0.89	72.5	68.0	−4.6
Caschool	17.3	0	0.007	0.045	0.67	1.75	0.45	14.0	14.0	0.0
Fatality	17.0	0	0.027	0.031	0.22	0.99	0.58	91.4	84.2	−7.1
highway	17.3	0	0.074	0.103	0.66	1.77	0.45	53.8	53.8	0.0
Hoops	18.8	0	0.028	0.086	0.20	0.50	0.33	16.3	16.3	0.0
infant_mortality	23.1	0	0.038	0.063	1.26	6.86	0.84	54.3	51.4	−2.9
InsuranceVote	19.6	0	0.033	0.144	0.33	0.54	0.59	88.7	88.7	0.0
iris	20.4	0	0.023	0.043	0.24	0.28	0.49	94.7	89.3	−5.3
MedGPA	19.5	0	0.052	0.090	0.43	1.25	0.48	63.7	63.7	0.0
midwest	16.2	0	0.010	0.031	1.57	11.43	0.87	84.2	68.7	−15.6
avg	18.9	0	0.030	0.067	0.70	3.57	0.60	63.4	59.8	−3.5

Table 5.11: Best results of autoencoders for each data set using grid search.

of cases, the number of attributes, the number of classes, the class entropy and the parameters. For a new data set, we execute the error prediction model on a set of instances representing the properties of the data set and generated grid. We chose the parameters from the grid which produce the lowest $\Delta(m1, m2)$.

The model was 5 x 10 cross validated on 17 data sets, which were used in Table 5.3. Table 5.12 shows the results, which suggest that using predicted parameters return on average better results than using defaults parameters of the generator. Nevertheless, Wilcoxon signed-rank test at $\alpha = 0.05$ fails to reject the null hypothesis that difference between approaches is zero. The approach with predicted parameters works better for 10 out of 17 data sets. The results show that average `m2d1` is lower for almost 3 percentage points and average variance for `m2d1` score is lower for 4 percentage points.

We showed that predicting parameters has potential to improve the results. Our predictor was built on only 10 data sets. Additionally, the possible values of the parameters were limited. As grid search needs a lot of computational power, it is difficult to do it for a large number of data sets. Overall, our conclusion is that grid search for a specific problem produces the best results for that problem, second to the model trained on the results of grid search. Using the default parameters is worse than both these options.

	t[s]	=	m($\Delta mean$)	m(Δ *std*)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
annealing	21.5	0	0.027	0.010	0.38	1.10	0.68 ± 0.04	99.55 ± 0.74	67.96 ± 18.62	-31.60
balance-scale	11.5	0	0.010	0.010	0.05	0.25	0.57 ± 0.04	81.31 ± 3.95	75.27 ± 6.02	-6.04
breast-cancer	9.1	0	0.022	0.015	0.10	0.46	0.67 ± 0.04	69.66 ± 6.72	56.06 ± 12.79	-13.61
breast-cancer-wdbc	6.1	0	0.045	0.015	0.30	1.84	0.40 ± 0.04	96.09 ± 2.90	92.97 ± 3.24	-3.12
breast-cancer-wisconsin	7.8	0	0.027	0.055	0.28	0.74	0.93 ± 0.02	96.10 ± 1.78	92.60 ± 3.46	-3.50
bridges-version1	4.7	0	0.009	0.075	0.15	0.38	0.34 ± 0.09	64.86 ± 13.78	45.02 ± 16.93	-19.84
bridges-version2	5.0	0	0.000	0.000	0.00	0.00	0.37 ± 0.09	63.25 ± 11.11	53.02 ± 14.52	-10.23
dermatology	10.6	0	0.009	0.058	0.17	0.41	0.77 ± 0.06	96.71 ± 3.09	78.76 ± 9.74	-17.95
ecoli	8.1	0	0.007	0.066	0.34	0.51	0.90 ± 0.07	83.63 ± 5.73	75.01 ± 8.11	-8.62
flags	7.1	0	0.011	0.027	0.14	0.54	0.59 ± 0.03	62.84 ± 12.28	37.18 ± 10.73	-25.66
glass	7.9	0	0.021	0.036	0.63	3.42	0.88 ± 0.05	74.29 ± 9.62	38.46 ± 14.57	-35.83
haberman	8.1	0	0.027	0.076	0.24	0.66	0.42 ± 0.07	70.55 ± 8.30	66.41 ± 11.36	-4.14
iris	6.3	0	0.019	0.019	0.22	0.37	0.54 ± 0.07	94.83 ± 5.67	84.00 ± 13.65	-10.83
post-operative	5.9	0	0.000	0.000	0.00	0.00	0.32 ± 0.04	62.78 ± 15.22	56.11 ± 14.90	-6.67
primary-tumor	12.3	0	0.015	0.082	0.13	0.53	0.66 ± 0.05	40.83 ± 8.33	33.56 ± 8.58	-7.27
soybean-large	14.1	0	0.010	0.054	0.17	1.14	0.71 ± 0.04	89.93 ± 5.48	83.13 ± 17.66	-6.80
tic-tac-toe	8.8	0	0.000	0.000	0.00	0.00	0.36 ± 0.02	95.77 ± 2.31	55.56 ± 6.27	-40.22
avg	9.1	0	0.015	0.035	0.19	0.73	0.59 ± 0.20	79.00 ± 16.47	64.18 ± 18.61	-14.82

Table 5.12: Autoencoder results with parameters predicting based on grid search.

Chapter 6

Conclusion

The goal of the thesis was to develop a generator of semi-artificial data based on autoencoders, which would improve upon existing approaches. We developed generators based on autoencoders and variational autoencoders. We wanted that our solution is general and may be used on any data set, therefore we implemented dynamic autoencoders and dynamic variational autoencoders without any predefined structure.

The performance of the generators was measured with clustering and prediction performance. If original and generated data are similar, clustering and prediction should return similar results. Results show that autoencoder based generators produce better results than variational autoencoders on classification problems. Nevertheless, our generators are often inferior to RBF based generators [31]. Our generators perform best on data sets with a small number of mixed attributes and balanced classes. They perform better if more training instances are available.

We tried to a find good set of default parameters by testing each parameter separately. This approach assumes that parameters are independent from the specific dataset and from each other. Both assumptions are not true, therefore we used grid search to find an improved combination of parameters. We showed that grid search significantly improves the results. Since training neural networks requires a lot of computational power, extended grid search

is difficult to execute, as there are many combinations of parameters. A better solution is to implement a system that sets the parameters specifically for each data set. We showed that predicting parameters is feasible and produces similar results. The problem of this approach is producing enough training data, as the grid search is time-consuming. Currently, the loss function is hard-coded, and therefore, changing it to a parameter could improve the results. For example, mean squared error is more suitable for regression problems than cross-entropy.

The generators we implemented are not suitable for sequential or image data. To support these types of data we would need to develop new algorithms, e.q. LSTM-based autoencoders for sequential data and convolutional autoencoders for image data [9].

Bibliography

[1] House Sales in King County, USA, https://www.kaggle.com/harlfoxem/housesalesprediction, accessed: 26.05.2018 (2016).

[2] Diamonds data set, https://www.kaggle.com/shivam2503/diamonds, accessed: 26.05.2018 (2017).

[3] V. Arel-Bundock, R datasets, https://vincentarelbundock.github.io/Rdatasets/, accessed: 26.05.2018 (2018).

[4] Y. Bengio, Learning Deep Architectures for AI, Foundations and Trends in Machine Learning 2 (1) (2009) 1–127.

[5] N. Budincsevity, Weather in Szeged 2006-2016, https://www.kaggle.com/budincsevity/szeged-weather, accessed: 26.05.2018 (2017).

[6] D. Charte, F. Charte, S. García, M. J. del Jesus, F. Herrera, A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines, Information Fusion 44 (2018) 78 – 96.

[7] M. Choi, Medical Cost Personal Datasets, https://www.kaggle.com/mirichoi0218/insurance/data, accessed: 26.05.2018 (2018).

[8] F. Chollet, et al., Keras, https://keras.io (2015).

[9] F. Chollet, Building Autoencoders in Keras, https://blog.keras.io/building-autoencoders-in-keras.html, accessed: 24.05.2018.

[10] D. Dheeru, E. Karra Taniskidou, UCI Machine Learning Repository, http://archive.ics.uci.edu/ml (2017).

[11] C. Doersch, Tutorial on variational autoencoders, arXiv preprint arXiv:1606.05908 (2016).

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, in: Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, 2014, pp. 2672–2680.

[13] A. G. Gray, A. W. Moore, Nonparametric density estimation: Toward computational tractability, in: Proceedings of the 2003 SIAM International Conference on Data Mining, SIAM, 2003, pp. 203–211.

[14] K. Gregor, I. Danihelka, A. Graves, D. Rezende, D. Wierstra, DRAW: A Recurrent Neural Network For Image Generation, in: International Conference on Machine Learning, 2015, pp. 1462–1471.

[15] R. Hecht-Nielsen, Replicator Neural Networks for Universal Optimal Source Coding, Science 269 (5232) (1995) 1860–1863.

[16] J. Jordan, Variational autoencoders, https://www.jeremyjordan.me/variational-autoencoders/, accessed: 24.05.2018.

[17] A. Karpathy, Convolutional Neural Networks for Visual Recognition, http://cs231n.github.io/neural-networks-1/, accessed: 24.05.2018.

[18] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[19] D. P. Kingma, S. Mohamed, D. J. Rezende, M. Welling, Semi-supervised learning with deep generative models, in: Advances in Neural Information Processing Systems, 2014, pp. 3581–3589.

[20] D. P. Kingma, M. Welling, Auto-encoding Variational Bayes, arXiv preprint arXiv:1312.6114 (2013).

[21] M. A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AIChE Journal 37 (2) (1991) 233–243.

[22] T. D. Kulkarni, W. F. Whitney, P. Kohli, J. Tenenbaum, Deep convolutional inverse graphics network, in: Advances in Neural Information Processing Systems, 2015, pp. 2539–2547.

[23] J. Li, Honey production In The USA (1998-2012), `https://www.kaggle.com/jessicali9530/honey-production`, accessed: 26.05.2018 (2018).

[24] J. Li, T. Luong, D. Jurafsky, A Hierarchical Neural Autoencoder for Paragraphs and Documents, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Vol. 1, 2015, pp. 1106–1115.

[25] X. Lu, Y. Tsao, S. Matsuda, C. Hori, Speech enhancement based on deep denoising autoencoder, in: Interspeech, 2013, pp. 436–440.

[26] W. McKinney, et al., Data structures for statistical computing in Python, in: Proceedings of the 9th Python in Science Conference, Vol. 445, 2010, pp. 51–56.

[27] V. Miranda, J. Krstulović, H. Keko, C. Moreira, J. Pereira, Reconstructing missing data in state estimation with autoencoders, IEEE Transactions on Power Systems 27 (2) (2012) 604–611.

[28] T. E. Oliphant, A guide to NumPy, Vol. 1, Trelgol Publishing USA, 2006.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vander-

plas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[30] D. J. Rezende, S. Mohamed, D. Wierstra, Stochastic Backpropagation and Approximate Inference in Deep Generative Models, in: International Conference on Machine Learning, 2014, pp. 1278–1286.

[31] M. Robnik-Šikonja, Data generators for learning systems based on RBF networks, IEEE transactions on neural networks and learning systems 27 (5) (2016) 926–938.

[32] T. Salimans, D. Kingma, M. Welling, Markov chain Monte Carlo and variational inference: Bridging the gap, in: International Conference on Machine Learning, 2015, pp. 1218–1226.

[33] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (Supplement C) (2015) 85 – 117.

[34] H. Schwenk, Y. Bengio, Boosting Neural Networks, Neural Computation 12 (8) (2000) 1869–1887.

[35] S. Sheather, A modern approach to regression with R (data sets), `http://www.stat.tamu.edu/~sheather/book/data_sets.php`, accessed: 26.05.2018 (2009).

[36] K. Sohn, H. Lee, X. Yan, Learning structured output representation using deep conditional generative models, in: Advances in Neural Information Processing Systems, 2015, pp. 3483–3491.

[37] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, Coursera: Neural networks for machine learning 4 (2) (2012) 26–31.

[38] E. Tyantov, Deep Learning Achievements Over the Past Year, `https://blog.statsbot.co/deep-learning-achievements-4c563e034257`, accessed: 23.05.2018.

[39] J. Walker, C. Doersch, A. Gupta, M. Hebert, An uncertain future: Forecasting from static images using variational autoencoders, in: European Conference on Computer Vision, Springer, 2016, pp. 835–851.

Appendix A

Setting Default Parameters

The testing of generators to get default parameters was done on the following data sets:

- biomass
- Caschool
- Fatality
- Fishing
- highway
- Hoops
- infant_mortality
- InsuranceVote
- iris
- MedGPA
- midwest
- msleep
- Tobacco

The testing was done in same way as described in Section 4 with one difference – we used 3-fold cross-validation instead of 5x10-fold cross validation in order for testing to be less time consuming.

Below we present the following results:

- autoencoders depending on number of epochs in Table A.1
- VAEs depending on number of epochs in Table A.2

- autoencoders depending on the `r` parameter in Table A.3
- VAEs depending on the `r` parameter in Table A.4
- autoencoders depending on the drop rate in Table A.5
- VAEs depending on the drop rate in Table A.6
- autoencoders depending on the batch size in Table A.7
- VAEs depending on the batch size Table A.8
- autoencoders depending on the activation function in Table A.9
- VAEs depending on the activation function in Table A.10

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	$\Delta(m_1, m_2)$
epoch=5	5.1	0	0.125	0.078	1.12	5.23	0.44	66	40	26
epoch=10	5.1	0	0.096	0.073	0.94	4.86	0.50	66	42	24
epoch=20	5.7	0	0.076	0.061	0.80	4.36	0.59	67	49	18
epoch=50	7.6	0	0.067	0.060	0.78	4.62	0.65	66	51	15
epoch=100	9.0	0	0.064	0.053	0.78	4.77	0.67	67	56	11

Table A.1: Comparison of the autoencoder results depending on the epoch.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
epoch=25	5.2	0	0.201	0.076	1.55	5.97	0.32	67	40	27
epoch=50	6.3	0	0.193	0.084	1.53	5.99	0.36	66	45	21
epoch=100	6.3	0	0.190	0.082	1.50	5.62	0.33	67	40	27
epoch=250	6.5	0	0.188	0.084	1.50	5.93	0.38	67	42	25
epoch=500	6.4	0	0.189	0.083	1.50	5.90	0.38	67	38	29

Table A.2: Comparison of the VAE results depending on the epoch.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
r=0.1	4.1	0	0.103	0.094	1.00	5.11	0.39	67	39	28
r=0.2	4.2	0	0.102	0.087	0.98	5.19	0.46	67	39	28
r=0.3	4.3	0	0.114	0.079	1.03	4.87	0.50	66	45	21
r=0.4	4.4	0	0.105	0.076	1.01	5.17	0.50	66	43	23
r=0.5	4.7	0	0.097	0.069	0.96	5.05	0.49	67	43	24
r=0.6	5.2	0	0.095	0.068	0.89	4.42	0.51	66	44	22
r=0.7	5.1	0	0.099	0.060	0.94	4.79	0.52	68	46	22
r=0.8	5.0	0	0.093	0.069	0.95	4.99	0.52	66	45	21
r=0.9	5.3	0	0.092	0.061	0.88	4.52	0.53	69	44	25

Table A.3: Comparison of the autoencoder results depending on the r parameter.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
r=0.1	5.7	0	0.189	0.083	1.55	6.22	0.37	68	43	25
r=0.2	5.8	0	0.193	0.084	1.53	6.03	0.36	68	38	30
r=0.3	5.9	0	0.191	0.084	1.49	5.79	0.34	66	39	27
r=0.4	5.7	0	0.193	0.081	1.53	5.99	0.36	68	39	29
r=0.5	5.9	0	0.189	0.086	1.55	6.28	0.36	67	37	30
r=0.6	5.7	0	0.192	0.082	1.52	5.83	0.36	67	39	28
r=0.7	5.7	0	0.190	0.081	1.55	6.09	0.33	68	42	26
r=0.8	5.6	0	0.196	0.081	1.55	6.12	0.36	67	42	25
r=0.9	5.7	0	0.195	0.082	1.54	5.92	0.37	68	41	26

Table A.4: Comparison of the VAE results depending on the r parameter.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
drop=0.0	4.9	0	0.095	0.072	0.94	4.68	0.49	68	46	22
drop=0.1	5.9	0	0.102	0.073	0.93	4.37	0.51	67	41	26
drop=0.2	5.4	0	0.102	0.076	1.00	4.74	0.48	68	43	25
drop=0.3	5.8	0	0.097	0.077	1.04	4.80	0.47	67	40	27
drop=0.4	5.0	0	0.112	0.079	1.04	4.64	0.49	69	41	27
drop=0.5	4.8	0	0.117	0.076	1.09	4.82	0.49	67	42	25

Table A.5: Comparison of the autoencoder results depending on the drop rate parameter.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
drop=0.0	5.6	0	0.192	0.083	1.54	5.82	0.32	67	42	25
drop=0.1	6.9	0	0.190	0.085	1.52	5.78	0.36	66	40	26
drop=0.2	6.6	0	0.186	0.088	1.53	5.77	0.36	67	42	25
drop=0.3	6.5	0	0.187	0.091	1.52	5.82	0.35	67	40	28
drop=0.4	6.4	0	0.183	0.091	1.52	5.96	0.34	69	38	31
drop=0.5	6.4	0	0.180	0.094	1.54	6.07	0.35	67	39	28

Table A.6: Comparison of the VAE results depending on the drop rate parameter.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
batch=16	4.9	0	0.101	0.072	0.95	4.81	0.52	70	46	24
batch=32	4.5	0	0.128	0.081	1.14	5.21	0.44	67	40	27
batch=64	4.3	0	0.133	0.082	1.03	4.38	0.43	67	36	31
batch=128	4.1	0	0.156	0.073	1.25	5.39	0.35	68	34	34
batch=256	4.0	0	0.182	0.070	1.38	5.72	0.33	68	38	30

Table A.7: Comparison of the autoencoder results depending on the batch size.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
batch=16	7.8	0	0.175	0.085	1.49	6.02	0.34	68	45	23
batch=32	6.2	0	0.191	0.082	1.51	5.85	0.36	69	41	28
batch=64	5.4	0	0.201	0.082	1.49	5.39	0.31	66	42	24
batch=128	5.1	0	0.209	0.078	1.58	5.99	0.32	68	42	25
batch=256	4.8	0	0.216	0.074	1.58	6.03	0.29	67	37	29

Table A.8: Comparison of the VAE results depending on the batch size.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
act=relu(epoch=10)	4.8	0	0.094	0.070	1.16	5.04	0.49	67	41	26
act=relu(epoch=100)	7.1	0	0.020	0.033	0.56	3.39	0.71	66	59	7
act=tanh(epoch=10)	5.3	0	0.100	0.078	0.94	4.73	0.51	67	42	24
act=tanh(epoch=100)	8.4	0	0.061	0.056	0.80	4.84	0.67	65	55	10

Table A.9: Comparison of the autoencoder results depending on the activation function.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
act=relu(epoch=10)	6.2	0	0.163	0.088	1.52	5.68	0.44	67	38	29
act=relu(epoch=100)	5.9	0	0.170	0.089	1.54	6.01	0.40	68	37	30
act=tanh(epoch=10)	6.3	0	0.193	0.083	1.51	5.82	0.35	68	41	26
act=tanh(epoch=100)	5.9	0	0.193	0.082	1.52	5.93	0.37	66	42	25

Table A.10: Comparison of the VAE results depending on the activation function.

Appendix B

Choosing Input for Generators

Different inputs can be used to get generated data as an output. The testing was done on the same data sets and in same way as described in Section A.

B.1 Generators Based on Autoencoders

Table B.1 presents the results with different methods of generating data, where:

- kde – generating data using kde, without neural networks
- kde_beg – using samples from kde for numerical attributes and samples from original distrubition of categories for categorial attributes as an input of an autoencoder
- kde_mid – encoding the original data using an encoder, sample from encoded data using kde and using the samples as an input for a decoder
- norm_beg – using samples from $\mathcal{N}(0, 1)$ for numerical attributes and samples from original distrubition of categories for categorial attributes as an input of an autoencoder
- norm_mid – using samples from $\mathcal{N}(0, 1)$ as an input of a decoder

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
kde	0.0	0	0.024	0.836	1.64	5.59	0.27	66.7	35.2	−31.6
kde_beg	5.8	0	0.046	0.086	0.80	4.79	0.63	66.0	54.0	−12.0
kde_mid	5.8	0	0.022	0.042	0.53	3.47	0.73	68.8	58.0	−10.9
norm_beg	5.8	0	0.022	0.074	0.83	4.26	0.71	66.5	51.4	−15.1
norm_mid	5.3	0	0.168	0.059	1.56	5.75	0.57	68.6	43.9	−24.7
unif_beg	6.7	0	0.136	0.065	1.36	5.85	0.65	65.6	46.5	−19.2
unif_mid	5.8	0	0.185	0.084	1.65	5.42	0.50	68.1	41.8	−26.3

Table B.1: Comparison of different inputs for generating data with autoencoders.

	t[s]	=	m($\Delta mean$)	m(Δstd)	m($\Delta\gamma_1$)	m($\Delta\gamma_2$)	ARI	m1d1[%]	m2d1[%]	Δm
norm	6.5	0	0.180	0.070	1.50	6.16	0.38	67.5	44.3	−23.2
unif01	6.4	0	0.189	0.130	1.57	5.55	0.28	67.1	35.6	−31.5
unif	6.4	0	0.167	0.095	1.48	5.87	0.36	67.3	43.1	−24.2

Table B.2: Comparison of different inputs for generating data with variational autoencoders.

- unif_beg – using samples from $\mathcal{U}(0, 1)$ for numerical attributes and samples from original distrubition of categories for categorial attributes as an input of an autoencoder
- unif_mid – using samples from $\mathcal{U}(0, 1)$ as an input of a decoder

B.2 Generators Based on VAEs

Table B.2 presents the results with different methods of generating data, where:

- norm – using samples from $\mathcal{N}(0, 1)$ as an input of a decoder
- unif01 – using samples from $\mathcal{U}(0, 1)$ as an input of a decoder
- unif – using samples from $\mathcal{U}(-1, 1)$ as an input of a decoder