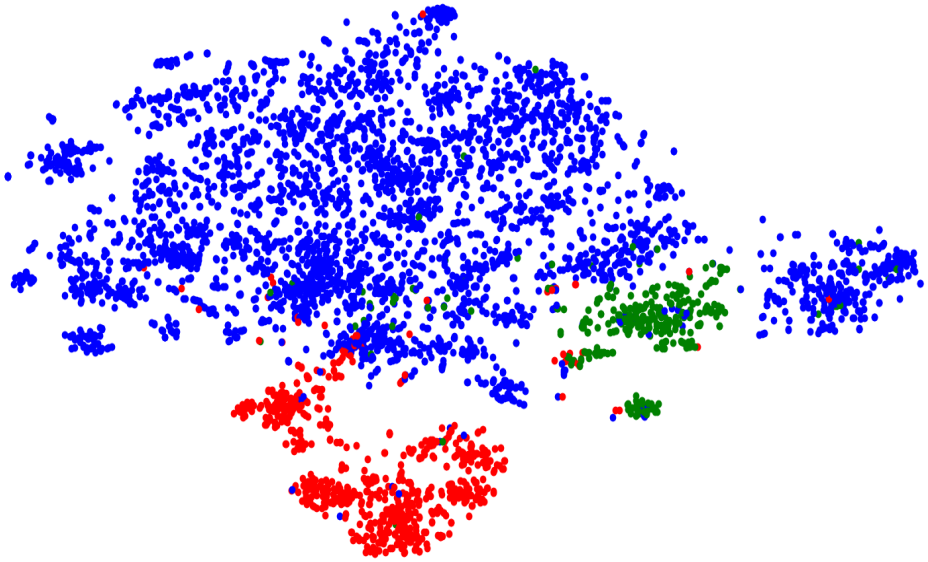
CHALMERS
UNIVERSITY OF TECHNOLOGY

Machine Learning for Prediction of Antibiotic Resistance

A Neural Network based model for Prediction of Minimum Inhibitory Concentration for Nontyphoidal *Salmonella*



Master's thesis in Engineering Mathematics and Computational Science

Emil Carlsson
Department of Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Machine Learning for Prediction of Antibiotic Resistance

Emil Carlsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Science
Chalmers University of Technology
Gothenburg, Sweden 2019

Machine Learning for Prediction of Antibiotic Resistance
Emil Carlsson

Supervisor: Erik Dyrkell, 1928 Diagnostics
Examiner: Torbjörn Lundh, Department of Mathematics

Department of Mathematics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Machine Learning for Prediction of Antibiotic Resistance
A Neural Network based model for Prediction of Minimum Inhibitory Concentration for Nontyphoidal *Salmonella*
Emil Carlsson
Department of Mathematics
Chalmers University of Technology

Abstract

This thesis aims to investigate whether machine learning can be used to diagnose whether a bacteria is resistance towards a certain antibiotic or not. This will be done by building a prediction model for prediction of minimum inhibitory concentration. Minimum inhibitory concentration is defined as the minimum dosage of a drug needed in order to inhibit a infection or disease.

To do this, a labeled dataset consisting of 4964 genomes from Salmonella bacteria with corresponding minimum inhibitory concentrations for up to 15 antibiotics where used alongside a unlabeled dataset of Salmonella genomes taken from *ncbi GenBank*.

Further, due to the small size of the dataset compared to the length of a Salmonella genome, more than 4 000 000 nucleotides, we divided each genome into k-mers and viewed each k-mer as a word. The genome can then be viewed as a document and the problem at hand becomes to classify this document w.r.t antibiotic resistance. To classify this document we took a bag-of-word approach, counting the occurrence of each k-mer and then producing a vector based on the count for each genome. The bag-of-word approach resulted in an information loss regarding the context of certain k-mers but made further processing feasible.

Furthermore, we considered two different machine learning model for the given task. A standard feedforward neural network trained in a supervised setting and a ladder network trained in a semi-supervised setting. We trained the networks for prediction of inhibitory concentration for all the 15 antibiotics simultaneously. To handle missing labels in the data we constructed a customized output layer consisting of 15 softmax layers concatenated. Given a missing label we simply ignored to gradient from the corresponding softmax layer. The training set was also over-sampled using two different techniques based on bootstrapping and synthetic minority over-sampling.

Moreover, it was found, through hyperparemeter tuning using the Parzen Tree estimator, that the semi-supervised learning did not improve the accuracy and a standard feedforward neural network had the best accuracy when it came to predicting exact minimum inhibitory concentration. Our feedforward neural network was then compared to baseline model, which was based on the distribution of labels in the dataset, and an already existing machine learning model trained on the considered dataset.

It was found that our feedforward neural network outperformed both these models when it comes to prediction minimum inhibitory concentrations. The average accuracy for prediction of exact minimum inhibitory concentration where 0.78 and when the result was translated to the labels sensitive, intermediate and resistance towards an antibiotic the model got an average accuracy of 0.97.

In addition, we evaluated our model with respect to the error rates defined by the *National Antimicrobial Resistance Monitoring System* and the error rates where found to not be low enough to be used in a clinical setting. We think that this is a combination of the limitations with a bag-of-word approach and the lack of data.

Nevertheless, from this work we can conclude that machine learning is an intresting and prominent approach to autonomous prediction of of minimum inhibitory concentration and diagnosis of antibiotic resistance. However, several problems like the interpretability of the models and skewness in the datasets are yet to be solved before a machine learning model can be used on a clinical setting for this purpose.

We end this thesis with a discussion regarding future work that could solve many of the problems encountered throughout this thesis.

Keywords: Machine Learning, Salmonella, Antibiotic Resistance, Minimum Inhibitory Concentration, Neural Network, Ladder Network, Bayesian Optimization

Acknowledgements

I want to thank my supervisors Torbjörn Lundh and Fredrik Dyrkell for enthusiastic support throughout this thesis work. I would also like to thank the entire team at 1928 Diagnostics for all the feedback and interesting discussions during my time at the company.

Emil Carlsson, Gothenburg, June 2019

Contents

1
Introduction

Antibiotic resistance is, alongside the climate changes and lack of freshwater, the most serious threat against mankind according to WHO [1]. Alexander Flemming, who found the pencillin in 1928, predicted that mal-use and over usage of antibiotics would lead to a post-antibiotic era where minor infections can no longer be treated and become lethal[2]. This era is approaching, already now there are pan resistant superbugs present in the population, and we need new technology in order to halt this negative development.

To successfully treat an resistant infection and to control the spread, fast decisions have to be made when it comes to diagnosis and treatments. However, to diagnose antibiotic resistance today one often cultivate the bacteria on agar-plates and then expose it to different antibiotics. In many cases this is a very time-consuming process and inadequate when fast decisions must be made. For example, the bacteria *Tubercolosis* takes about two weeks to cultivate on a agar-plate [3]. Going two weeks with no or wrong treatment might be devastating for a patient. Hence, there is a need for faster methods when it comes to antibiotic resistance profiling and virulence factor detection.

1.1 1928 Diagnostics

1928 Diagnostics is a Gothenburg based start-up tackling the growing problem of antibiotic resistance. The company provides a cloud-based platform which hospitals can upload the genome of a bacteria in order to get antibiotic resistance profile of the bacteria. By matching the digital representation of the genome with other genomes in a curated database can 1928 Diagnostics provide fast classification and analysis of bacteria genomes. This enables faster diagnosis and a more rigorous infection tracing then before possible at hospitals.

1.2 Antibiotic Resistance and Machine Learning

Many of the more prominent algorithmic methods regarding antibiotic resistance detection are based matching towards a database, like 1928 Diagnostics is doing. An other example is ResFinder [4]. These methods can be said to be "memory" based since they mimic the human trait memorization. This becomes a limitation since, unlike the human mind, these methods cannot learn and generalize from their memories. It would be a great advantageous if the methods instead where able to find patterns and make general conclusion based on their memories.

Machine learning is a field within computer science focusing on developing algorithms able to perform advanced tasks without being explicit programmed to do so. This is done by letting mathematical models extract knowledge from datasets and can be viewed as the opposite of algorithms matching towards a database. Instead of just memorizing the data, a machine learning algorithm tries to find patterns within the data and in the extension gain some general knowledge based these patterns [5, p. 1-5].

Moreover, several approaches using machine learning in order to predict antibiotics have lately been made. A very prominent approach where made by Daniel Veltri et alt. in the paper *Deep Learning improves antimicrobial peptide recognition* [6]. The authors present a sequence based machine learning model, using both convolutional layers and embedding layers. The model is trained on 1424 peptide sequences with a length varying from 10 to 150 aminoacids. The authors shows that the model outperforms several state-of-the-art classification algorithms. However, the dataset used is quite small and tests performed on larger datasets would be intresting in order to investigate the robustness of the algorithm further. The length of the sequences trained on are also quite small in comparison to the raw output from different sequencing methods. Thus, in order to use this method on an entire bacteria genome some *a priori* information or pre-processing step is probably needed.

Furthermore, the paper *Using Machine Learning To Predict Antimicrobial MICs and Associated Genomic Features for Nontyphoidal Salmonella* by Marcus Nguyen et alt. presents a machine learning model trained on 4500 Salmonella genomes. This is, to our knowledge, the largest machine learning study performed for on whole-genome sequenced data for prediction of antibiotic resistance. The authors trains a extreme gradient boosted decision tree and are able to extract features correlated to resistance from the trained model. This makes the model somewhat interpretable which is important in a clinical setting.

In addition, the article *DeepARG: a deep learning approach for predicting antibiotic resistance genes for metagenomic data* by G. Arango-Argoty et alt. presents a method for prediction of resistance genes in genetic material directly taken from some environment [7]. This means that the model is able to take genetic material from several different species at the same time and detect antibiotic resistance for that set which require minimal pre-processing. The model is based representing an input as a vector of similarities between the input and known antibiotic resistance genes. The similarity vector is then feed into a machine learning model which tries to classify the input.

2
Aims

This thesis aims to investigate how machine learning and especially neural networks can be used in order to predict antibiotic resistance for the bacteria *Salmonella* using DNA-sequences as input. The thesis can be divided into three parts

1. Developing a pre-processing step that facilitates learning for machine learning models.
2. Constructing a prediction model based on neural networks
3. Investigating whether it is possible to improve the prediction model using semi-supervised learning

3
Theory

In this chapter we present the theory needed in order to understand the methods used and conclusions in this thesis. We assume the reader to be familiar with multi variable calculus, linear algebra and optimization.

3.1 DNA and Antibiotic Resistance

In the following section we present the biological theory needed to follow the reasoning throughout this thesis. We start by introducing DNA, genetic mechanisms and NGS. We then present more problem specific theory such as antibiotic resistance and minimum inhibitory concentration.

3.1.1 DNA

Offsprings tends to have similar traits as their parents. This is called *genetic inheritance* and plays a keyrole in evolution. The inheritance is caused by *genes* being transferred from the parent to the offspring. The total genetic material in an organism is referred to as the *genome* [9].
Further, genes are composed of a molecule called *deoxyribonucleic acid* (DNA) which is shown in figure 3.1. It is the DNA molecule who carries all the genetic information of an organism and this is encoded by base pairs. Each base pair consists of two *nucleotides*.
Furthermore, there exist four different nucleotides in the DNA : *cytosine* (C), *gaunine* (G), *adenine* (A) and *thymine* (T). These nucleotides forms the basis of the biological programming language [10].

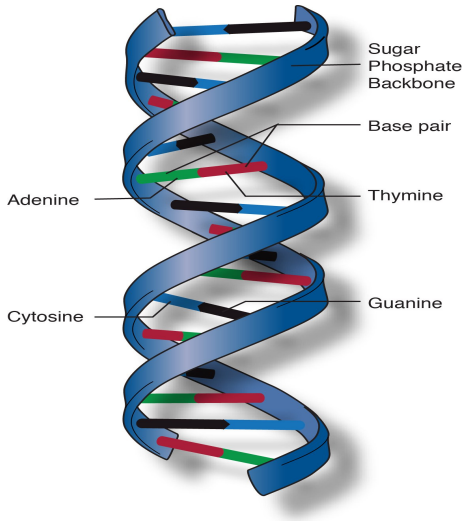


Figure 3.1: The DNA double helix. Figure taken from PlosOne with a CC BY license [8].

Moreover, most processes in an organism is in need of *proteins*. To construct a protein, the DNA is used as a instruction book. A sequence of nucleotides can be viewed as a sequences of operation which leads up to the construction of a protein. To be more precise, a triplet of succeeding nucleotides forms a *codon*. Each codon encodes an *amino acid* and a chain of amino acids forms a protein [9].

However, it might happen that parts of the DNA changes and this is called a *mutation*. For example, certain parts can be deleted, a single nucleotide might change or an entire gene can be added. This might in the extension change functions within the bacteria.
Mutations serves as a fuel for evolution. A mutation can be devastating and have a deadly outcome for the organism. On the other hand can a mutation, in rare cases, provide a significant advantages compared to other organisms. For example by making a bacteria resistant towards antibiotics.
A certain state of the DNA molecule is referred to as *genotype* and the resulting effect is referred to as *phenotype* [11]. An example of this is a mutation causing antibiotic resistance. The mutation changes the genotype which makes the bacteria resistance, i.e the phenotype has changed.

3.1.2 Analyzing DNA

The DNA encodes most of the processes within an organism. Investigating and analyzing DNA is therefore interesting in order to, for examples, treat diseases or find out whether a bacteria is resistance or not towards a certain antibiotic.
However, in order to make rigorous analysis we would like to extract the information within the DNA molecule somwhow. In recent year it has become much easier to extract this information and get a digital representation of the DNA by using something called *Next Generation Sequencing* (NGS) [12].
NGS is a collection of techniques that first breaks down the genome in smaller fragments, so called *reads*. It is then possible to make digital representations of these reads based on the letters A, C, G, T [12]. These fragments are then pieced together in a process called *assembling*. In many cases one is not able the retrieve the entire DNA molecule through an assembling and one instead get several longer sequences called *contigs*. Contigs may cover one or several genes and are therefore often sufficient when it comes to analyzing single genes.

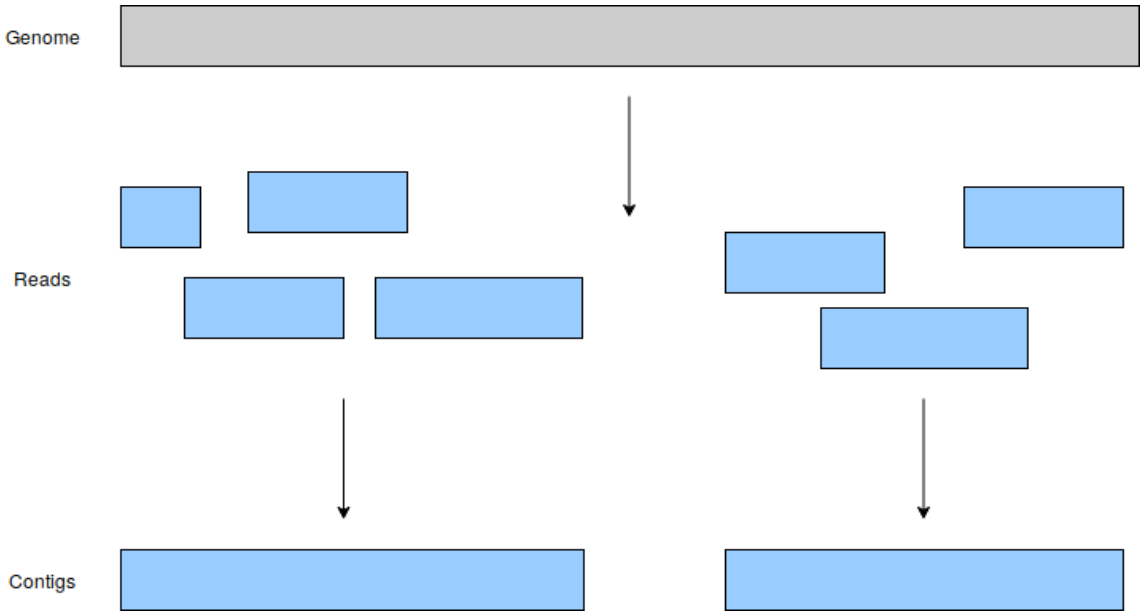


Figure 3.2: Illustration of the assembling process of a genome.

In addition, *k-mers* are a common tool in the field of genomics. A k-mer is substring of length k and given k there exist 4^k different k-mers. Hence, a k-mer becomes a sub-sequence of nucleotides and one often considers overlapping k-mers. In table 3.1 we illustrate how to compute overlapping 3-mers of a DNA-sequence.

DNA-sequence:			AGATTCAA			
3-mers:	AGA	GAT	ATT	TTC	TCA	CAA

Table 3.1: Illustration of how overlapping k-mers are calculated.

Computing the k-mers of a DNA-sequence does not require much computational power and provides a signature of the DNA-sequence. For example, given two individuals from the same species we can consider the distribution of k-mers in each genome to roughly estimate how similar the individuals are. If we in the extreme considers k-mers of the same size as the entire genome, all individuals will have its own unique k-mer. Therefore, by increasing k we increase to number of possible k-mers which in turn makes it easier to distinguish between different individuals, but this comes to an increased computational cost.

3.1.3 Antibiotic Resistance

Antibiotics are drugs used for treatment of infections caused by bacteria. Since Alexander Flemings discovery of penicillin in year 1928, and the dawn of the antibiotic era, countless of lives have been saved by antibiotics and many, before deadly, diseases have become treatable [13].

Moreover, like some humans are naturally more resistant towards certain diseases than others, individual bacteria within a species may be more resistant towards an antibiotic compared to the average bacteria within that species. When an antibiotic

is used against a bacterial infection, those bacteria not resistant will die while the resistant bacteria may thrive and potentially produce offspring that also carriers the resistant phenotype. This drives the natural selection within the infection. An over usage and misuse of an antibiotic in large scale might therefore lead to a resistant strain taking root in a society [14].

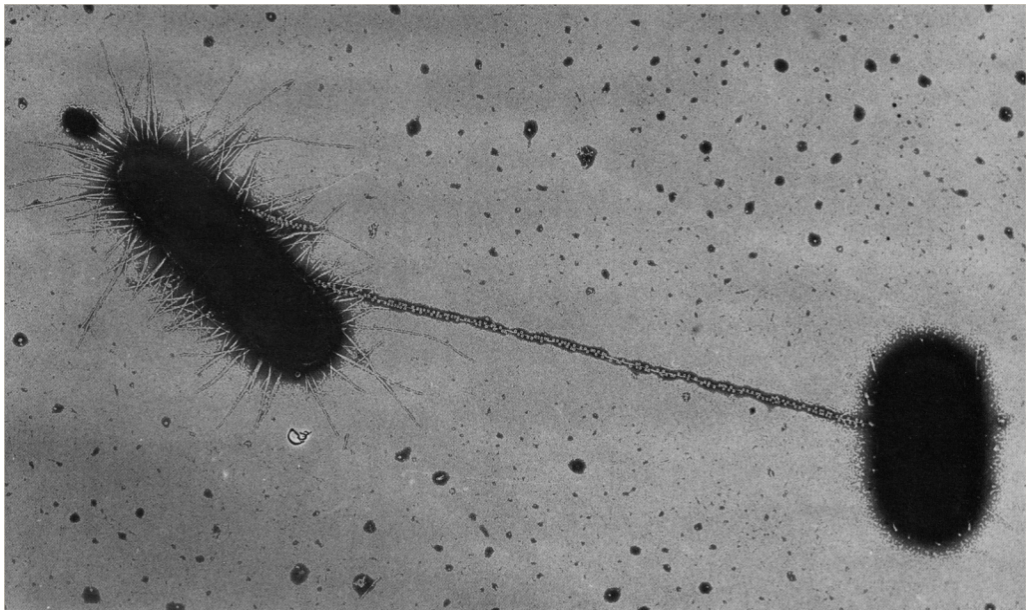


Figure 3.3: Picture of horizontial gene transfer. Courtesy of Charles C. Brinton Jr [15].

Furthermore, resistance may not only be spread by bacteria producing offspring. It is also possible for a bacteria to transfer DNA to to another bacteria. This is called *horizontial gene transfer* and makes it possible for one species of bacteria to pass a gene linked to antibiotic resistance to a different species.

3.1.3.1 Minimum Inhibitory Concentration

Minimum Inhibitory Concentration, or MIC-value, is defined as the lowest concentration of a drug($\mu g/mL$) such that the growth of the bacteria is inhibited [16]. Hence, the MIC-value becomes a continuous measure of how resistant a bacteria is towards a particular antibiotic and therefore suitable when it comes to optimizing treatment.

Further, MIC-values are often expressed as a two-fold dilution. This means that MIC-values often occurs on the form $2^n, n \in \boldsymbol{Z}$.

However, MIC-values of different antibiotics are not directly comparable. This since antibiotics might have different recommended maximum dosage or different biological mechanisms. Instead, control authorities like the U.S Food and Drug Administration (FDA) distributes guidelines from *National Antimicrobial Resistance Monitoring System* (NARMS) for how to translate a MIC-value to one of the labels, sensitive (S), intermediate (I) or resistance (R) based on how large the MIC-value is and the antibiotics of interest.

We present the guidelines used throughout this thesis for translating MIC-values into their corresponding labels for the bacteria *Salmonella enterica* in table 3.2. Note that all values are expressed on a two-fold dilution level, this is in line with FDAs guidelines for susceptibility tests [17].

	MIC Breakpoints (μg/L)		
Antibiotic	R$\geq$	I	S$\leq$
Aminoglycosides			
Gentamicin(GEN)	16	8	4
Kanamycin(KAN)	64	32	16
Streptomycin(STR)	64	-	32
Penicillins			
Amoxicillin(AUG)	32	16	8
Ampicillin(AMP)	32	16	8
Macrolides			
Azithromycin(AZI)	32	-	16
Cephems			
Cefoxitin(FOX)	32	16	8
Ceftriaxone(AXO)	4	2	1
Cefttifur(TIO)	8	4	2
Folate Pathway Inhibitors			
Sulfidoxazole(STR)	512	-	256
Trimethoprim(COT)	4	-	2
Phenicols			
Chloramphenicol(CHL)	32	16	8
Quinolones			
Ciprofloaxin(CIP)	4	2	1
Nalidixic acid(NAL)	32	-	16
Tetracyclines			
Tetracycline(TET)	16	8	4

Table 3.2: MIC-value thresholds for the different antibiotics considered throughout this thesis[17]

3.2 Machine Learning

Machine learning is branch of artificial intelligence which aims to design algorithms that can perform specific tasks without explicit instructions [5]. This is achieved by a data-driven approach where the algorithm learns a mathematical model from a training dataset.

To give a more mathematical description, consider a function

$$g : \boldsymbol{X} \to \boldsymbol{Y}. \tag{3.1}$$

For example let $\boldsymbol{X}$ be the set of all possible genomes of a bacteria and $\boldsymbol{Y}$ the space of MIC-values. Then g is the biological function determining the MIC-value for each antibiotic given a bacteria genome $\boldsymbol{x}$. Our goal in machine learning is to learn the function g, but in practise this is impossible since we know nothing about the functional space that g lies in. Instead, we try to find a function in some functional space, $f \in F$, that serves as a good approximation of g. Hence, we would like to solve the optimization problem

$$\min_{f \in F} ||f - g||. \tag{3.2}$$

Unfortunately, this optimization problem is intractable since we do not know anything about g.

However, assume that we have a set of genomes $S \subset \boldsymbol{X}$ for which be have observed the corresponding MIC-values, i.e $g(S) \subset \mathbf{Y}$. For a given functional space F we could then solve

$$\min_{f \in F} \sum_{\boldsymbol{x} \in S} ||f(\boldsymbol{x}) - g(\boldsymbol{x})||. \tag{3.3}$$

Our optimization - or learning-problem thus have two phases. Phase 1 is picking the space F this is equivalent to picking a model and its hyperparameters. In phase 2 we train our chosen model, i.e finding the optimal $f \in F$. Optimization over a set S where $g(S)$ is known is called *supervised learning* within the field of machine learning.

Further, we can also perform *unsupervised learning*. In unsupervised learning, we search for structures and correlations in a dataset U but in contrary to the supervised learning we have no observation $g(U)$ to work with. Instead, we perform clustering and feature extraction [18]

Furthermore, one can combine the ideas behind supervised- and unsupervised learning into *semi-supervised learning*. Consider the two sets $S, U \subset \mathbf{X}$ where $g(S)$ is known and $g(U)$ is unknown. We try the utilize the dataset U when learning g by considering the following minimization problem

$$\min_{f \in F, f^* \in F^*} C_S \sum_{\boldsymbol{x}} \in S||f(\mathbf{x}) - g(\mathbf{x})|| + C_U \sum_{\boldsymbol{x} \in U} ||\mathbf{x} - f^*(f(\mathbf{x}))||, \tag{3.4}$$

where F^* is some space of functions satisfying $f^* : \mathbf{Y} \to \mathbf{X}$. The first term is the standard supervised learning presented in equation 3.3, in the last term we first use our machine learning model, or function, f to predict a vector in $\mathbf{Y}$. We then use this vector as an input to another machine learning model f^* trying to reconstruct the input to the first model f. Tuning the scaling factors C_S and C_U is crucial for this learning to succeed. Consider the example with MIC-values, it is reasonable to assume that the genome have a lot of influence on the MIC-values, we should therefore have a relatively high C_S term. On the contrary, we cannot expect the reconstruct the entire genome very well from just knowing the MIC-values. This since the genome codes for many phenotypes of the bacteria and MIC-value is only one of them, thus the C_U term should be low in relation to C_S. One can think of equation 3.4 as the supervised learning guiding the unsupervised learning towards the "right" clusters and features.

In practise we are not bounded to optimize the norm as we do in equation 3.3 and 3.4 and using other functions, like cross-entropy, might be more beneficial and computationally tractable. Thus, our optimization problem in a supervised setting becomes

$$\min_{f \in F} L_S(f(\mathbf{x}), \mathbf{y}) \tag{3.5}$$

and in a semi-supervised setting

$$\min_{f \in F} L_S(f(\mathbf{x}), \mathbf{y}) + L_U(f^*(f(\mathbf{x})), \mathbf{x}), \tag{3.6}$$

where $L_S(.,.)$ and $L_U(.,.)$ are arbitrary *loss functions* measuring the difference between the predicted outputs and the true outputs.

3.2.1 Neural Network

We define a neural network as a graph $G = (V, E)$, where V is a set of nodes and E a set of edges. Each node $n_i \in V$ has a state $h_i \in \mathbf{R}$ and a bias $\theta_i \in \mathbf{R}$ and each edge e_{ij} has a weight $w_{ij} \in \mathbf{R}$. We will refer to the nodes in a neural network as *neurons*.

The state of a neuron n_j is determined by the states of the neighbouring neurons in the following way

$$h_j = \sigma(\sum w_{ij} h_i + \theta_j), \tag{3.7}$$

where $\sigma(.)$ is a non-linear function referred to as the *activation function*. Hence, the neural network becomes a dynamical system. Equation 3.7 can also be expressed on matrix-form

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{h} + \theta), \tag{3.8}$$

where σ is applied element-wise.

In order to make use of a neural network we require that the dynamical system converges. However, it easy to see that the above definition along with equation 3.7 does not ensure convergence. Consider a network of two neurons with initial states of $h_1 = h_2 = 1$ and the following parameters $w_{1,2} = w_{2,1} = \theta_1 = \theta_2 = 1$ with an activation of $\sigma(x) = x^2$. The states of the neurons would tend to infinity.

However, one can ensure convergence of a neural network by considering a special type of neural network called feedforward neural network.

3.2.1.1 Feedforward Neural Network

Consider a neural network having the form of a directed acyclic graph. We initialize the top layer of the acyclic graph as our input vector and interpret the bottom layer as the output of the neural network, see figure 3.4, we can then ensure convergence of the neural network. This is called a feedforward neural network [18, pp. 165-172].

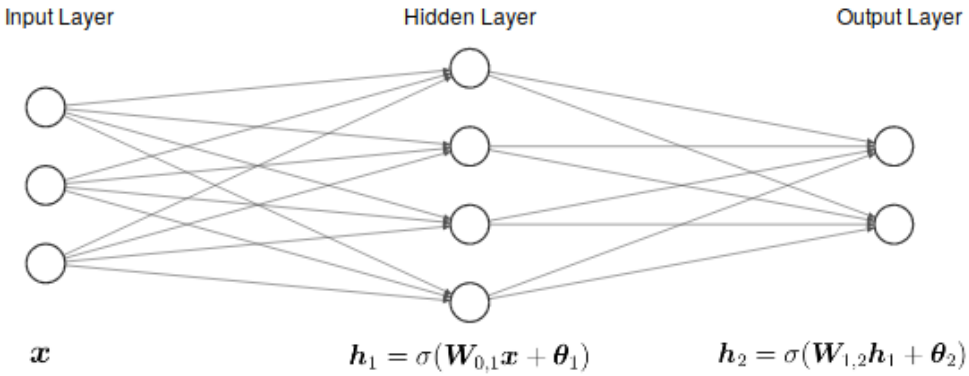


Figure 3.4: A feedfoward neural network with one hidden layer

Further, we denote an output-node in the feedforward neural network as O_i and treat this as a dimension in our output space $\mathbf{Y}$. Thus, we can interpret the feedforward neural network as a function:

$$f_{\mathbf{w},\boldsymbol{\theta}} : \mathbf{X} \to \mathbf{Y}, \tag{3.9}$$

where each layer, l, of the acyclic graph can be interpreted as a function

$$f^l_{\mathbf{w},\boldsymbol{\theta}} = \sigma(\mathbf{W_{l-1,l}}\mathbf{h_{l-1}} + \boldsymbol{\theta_l}), \tag{3.10}$$

where $\mathbf{W_{l-1,l}}$ denotes the weights between layers l-1 and l in the network and $\boldsymbol{\theta_l}$ denotes the biases at layer l. Hence, a feedfoward neural network with L layers becomes a chain of functions going from the input space, through several latent spaces, and ending in the output space

$$f_{\mathbf{w},\boldsymbol{\theta}} = f^L_{\mathbf{w},\boldsymbol{\theta}}(f^{L-1}_{\mathbf{w},\boldsymbol{\theta}}(...f^1_{\mathbf{w},\boldsymbol{\theta}}(\boldsymbol{x}))). \tag{3.11}$$

The internal layers, or latent spaces, in the feedforward neural network are referred to as *hidden layers*. The parameters $\boldsymbol{W}_{l-1,l}$ and $\boldsymbol{\theta}_l$ becomes the parameters to learn for each layer l.

It can be shown that feedforward neural networks with one hidden layer can approximate any continuous function on a compact set using a sigmoid non-linearity as activation in the hidden layer [19]. There exists other results showing that the same is true for other non-linear activation functions [20]. This is known as the universal approximation theorem.

Approximating any continuous function arbitrary well is a strong property and a good argument for using feedforward neural networks in general. Assuming that antibiotic resistance, and especially MIC-values, follows some continuous function which takes the genome as input is however not a good assumption. Antibiotic resistance depends, among many things, of mutations in the genome of a bacteria. Hence MIC-values might depend on the present or absence of certain nucleotides or k-mers. Thus, treating the genome as a sequence of binary or integer variables might be a good approach, rather than as continuous variables.

Nevertheless, feedforward neural network might still be a good approach for building a prediction model for antibiotic resistance and especially MIC-values. Any

continuous function g can be approximated arbitrary well by a feedforward neural network, more formally $\forall g \in C([0,1]^n)$

$$\exists \{f^{(t)}\}_{t=0}^{\infty} \subset F: \quad \lim_{t\to\infty} ||f^{(t)} - g|| = 0, \tag{3.12}$$

where F is the set of all feedforward neural networks with one layer. Hence, F becomes dense in $C([0,1]^n)$.

Further, $C([0,1]^n)$ is dense in $L^p([0,1]^n)$ which implies that F is dense in $L^p([0,1]^n)$ [21]. Hence,

$$\exists \{f^{(t)}\}_{t=0}^{\infty} \subset F: \quad \lim_{t\to\infty} ||f^{(t)} - g||_p = 0, \quad \forall g \in L^p, p \in [1,\infty). \tag{3.13}$$

Now, assume that $\exists g : ||g||_p < \infty, p \in [1,\infty)$, only depending on the DNA of a bacteria, that perfectly determines the MIC-value. We observe that a DNA-sequence of maximum length m can be perfectly described by a one-hot encoding $x \in \{0,1\}^{4m}$, g can thus be described as a discrete function over the set $\{0,1\}^{4m}$. We can think of a step-function $\hat{g} \in L^p[0,1]^{4m}$ satisfying $\hat{g}(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \{0,1\}^{4m}$ and by 3.12 we know that F is dense in $L^p([0,1]^n)$ for any $n \in \boldsymbol{N}$ and $1 \leq p < \infty$ so there exists a sequence of feedforward neural networks $\{f^{(t)}\}_{t=0}^{\infty} \subset F$ satisfying

$$\lim_{t\to\infty} ||f^{(t)} - \hat{g}||_p = 0. \tag{3.14}$$

3.2.1.2 Training a Feedfoward Neural Network

Training a feedforward neural network done by using a *backpropagation* algorithm. Backpropagation algorithms recursively updates the weights and biases layerwise in a feedforward neural network by exploting the chain-rule and then applying a stochastic gradient descent method.

The fact that the network has the form of a acyclic graph gives that the derivative of the loss function with respect to some weights between two layers $l-1$ and l, $\frac{L}{\boldsymbol{W}_{l-1,l}}$, only depends on weights later in the network [18, pp. 200-208]. The updating rule for the weights and biases becomes

$$\boldsymbol{W}_{l-1,l}^{t+1} = \boldsymbol{W}_{l-1,l}^{t} - \boldsymbol{\eta}_{l-1,l}^{t} \odot \boldsymbol{\delta}_l^t (\boldsymbol{h}_{l-1}^t)^T \tag{3.15}$$

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \boldsymbol{\eta}_l^t \odot \boldsymbol{\delta}_l^t \tag{3.16}$$

We define $\boldsymbol{\delta}_l^t$ as

$$\boldsymbol{\delta}_L^t = \frac{\partial L}{\partial f_{\boldsymbol{w},\boldsymbol{\theta}}^t(\boldsymbol{x})} \odot \sigma'(\boldsymbol{z}_L^t) \tag{3.17}$$

$$\boldsymbol{\delta}_l^t = (\boldsymbol{W}_{l,l+1}^t)^T \boldsymbol{\delta}_{l+1}^t \odot \sigma'(\boldsymbol{z}_l^t), \quad l < L \tag{3.18}$$

Where $\boldsymbol{\eta}_{l-1,l}^t$ is the *learning rate* for each weight in $\boldsymbol{W}_{l-1,l}^t$ and where we denote the Hadamard product by $\odot$ and let $\boldsymbol{z}_l^t$ denote the latent state of the layer l before the activation function is applied, $\boldsymbol{z}_l^t = \boldsymbol{W}_{l-1,l}^t \boldsymbol{h}_{l-1}^t + \boldsymbol{\theta}_l^t$. For the full derivation of the backpropagation algorithm on matrix-form we refer the reader to appendix B.1.

Given an "easy" loss-function, L, the derivative $\frac{\partial L}{\partial \boldsymbol{f_{w,\theta}(x)}}$ becomes easy to compute. For a square-loss function

$$L(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) = \frac{1}{2}||\boldsymbol{y} - f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})||^2_{L_2}. \tag{3.19}$$

We have

$$\frac{\partial L}{\partial f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})} = -(\boldsymbol{y} - f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})) \tag{3.20}$$

and for a cross-entropy loss-function

$$L(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) = -\boldsymbol{y}^T \log(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})) \tag{3.21}$$

the resulting derivative becomes

$$\frac{\partial L}{\partial f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})} = -(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})^{-1})^T \boldsymbol{y}. \tag{3.22}$$

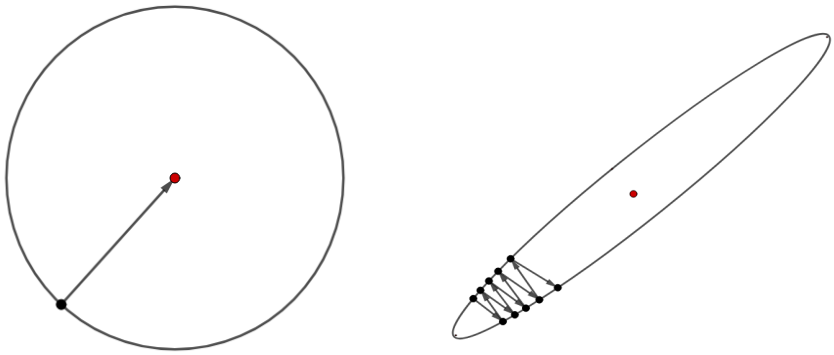
Moreover, when training a neural network with gradient descent we take the gradient with respect to the average loss over the entire training set S

$$\min_{\boldsymbol{w},\boldsymbol{\theta}} \frac{1}{|S|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in S} L(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}). \tag{3.23}$$

Stochastic Mini-batch Gradient Descent

Stochastic mini-batch gradient descent is extension of the classical gradient descent method where the training data is divided into randomly chosen small batches of size n for each epoch. The parameters are updated with the gradient descent scheme, equations 3.15 3.16, after each pass of a *mini-batch* through the network [18, p. 274-276].

To motivate the stochastic mini-batch gradient descent we notice that the loss-function can in many cases be locally approximated by a quadratic function given that the weights of the neural network is not to large [5, p. 237-239]. A problem can then occur with a crude gradient decent approach since the negative gradient will not point directly to a minimum if the level set does not form a sphere around the minimum. Given a too large learning rate, the algorithm can then start oscillating and in the extreme iterate between to states and never converge. We illustrate this for two dimensional level sets in figure 3.5.



(a) Gradient descent for a circular level set

(b) Gradient descent for an elliptic level set

Figure 3.5: Illustration of gradient descent on two different level sets. In (b) we can see how a too large learning rate can make the algorithm oscillate for an elliptic level set.

In order to break this oscillating behavior we use stochastic mini-batch gradient descent. Since the parameters will be updated after each pass of a mini-batch the resulting gradient used for updating may look different depending on the mini-batch used. Thus periodic oscillation between several states are not as possible as for a standard gradient descent algorithm.

The stochastic element for the optimization algorithm might also help to escape shallow local minima. This since a local minima for the standard gradient descent methods not necessarily must be a local minima for a mini-batch in the stochastic mini-batch gradient descent method. In figure 3.6 we illustrate how the stochastic behaviour helps breaking the oscillation behaviour of the standard gradient descent.

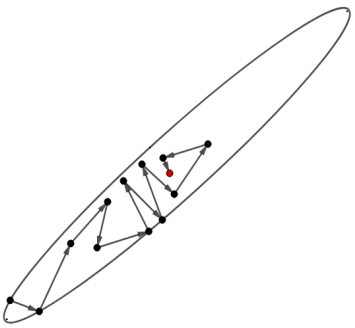


Figure 3.6: Stochastic mini-batch gradient descent on an elliptic level set.

Momentum

Further, we can make the training process more stable by introducing inertia in the parameter update by implementing a technique called *momentum* [22].

In the standard momentum method we accumulate the gradients from previous steps and adds this sum to the current updating term

$$\boldsymbol{W}_{l-1,l}^{t+1} = \boldsymbol{W}_{l-1,l}^{t} - (\boldsymbol{\eta}_{l-1,l}^{t} \odot \boldsymbol{\delta}_{l}^{t}(\boldsymbol{h}_{l-1}^{t})^{T} + \alpha \boldsymbol{M}_{l-1,l}^{t-1}) \tag{3.24}$$

$$\boldsymbol{M}_{l-1,l}^{t} = \boldsymbol{\eta}_{l-1,l}^{t} \odot \boldsymbol{\delta}_{l}^{t}(\boldsymbol{h}_{l-1}^{t})^{T} + \alpha \boldsymbol{M}_{l-1,l}^{t-1} \tag{3.25}$$

$$\boldsymbol{M}_{l-1,l}^{0} = \boldsymbol{0} \tag{3.26}$$

$$\boldsymbol{\theta}_{l}^{t+1} = \boldsymbol{\theta}_{l}^{t} - \boldsymbol{\eta}_{l-1,l}^{t} \odot \boldsymbol{\delta}_{l}^{t} - \alpha \boldsymbol{M}_{l}^{t-1} \tag{3.27}$$

$$\boldsymbol{M}_{l}^{t} = \boldsymbol{\eta}_{l}^{t} \odot \boldsymbol{\delta}_{l}^{t} + \alpha \boldsymbol{M}_{l}^{t-1} \tag{3.28}$$

$$\boldsymbol{M}_{l}^{0} = \boldsymbol{0} \tag{3.29}$$

by doing this, gradients pointing to towards opposite direction may cancel each other and we will dampen oscillations leading to a consensus direction, i.e the direction gradients for different mini-batches have in common [23].

Batch Normalization

Batch normalization is a technique used to increase stability and reduce training time for a neural network. Given a mini-batch of data B we propagate a data point $\boldsymbol{x}_i \in B$ through the feedforward neural network and achieve a latent state $\boldsymbol{z}_{l,i}$, before the activation function, at each layer $0 \leq l < L$. We normalize $\boldsymbol{z}_{l,i}$ with respect to the batch B as follows

$$\boldsymbol{\mu}_{l,B} = \frac{1}{|B|} \sum_{B} \boldsymbol{z}_{l,j} \tag{3.30}$$

$$\boldsymbol{\sigma}_{l,B} = \frac{1}{|B|} \sum_{B} (\boldsymbol{z}_{l,j} - \boldsymbol{\mu}_{l,B}) \odot (\boldsymbol{z}_{l,j} - \boldsymbol{\mu}_{l,B}) \tag{3.31}$$

$$\hat{\boldsymbol{z}}_{l,i} = \boldsymbol{\sigma}_{l,B}^{-1} \odot (\boldsymbol{z}_{l,i} - \boldsymbol{\mu}_{l,B}) \tag{3.32}$$

$$\bar{\boldsymbol{z}}_{l,i} = \boldsymbol{\gamma}_{l} \odot \hat{\boldsymbol{z}}_{l,i} + \boldsymbol{\beta}_{l} \tag{3.33}$$

where $\boldsymbol{\gamma}_l$ and $\boldsymbol{\beta}_l$ is a scale respective shift vector for layer l [24].

Moreover, we do batch normalization in order to reduce the internal *covariance shift* [24]. When training a neural network the parameters in one layer, l is updated each training iteration towards some local minima w.r.t the mini-batch considered but so are also the weights in all layers before l. Hence, the distribution of the input to layer l is dependent on the parameters in previous layers and will change when these parameters are updated. This can be problematic since we updated the parameters in layer l w.r.t. to one distribution of input data, but layer l will in next training iteration achieve another distribution of the input data. In the next training iteration the parameters in layer l will again be updated to compensate for this but so will the parameters in the previous layers as well, potentially causing the same issue again. This can slow done the training process a considerably amount [24].

If we instead perform batch normalization, we first normalize $\boldsymbol{z}_{l,i}$ w.r.t. to the current mini-batch. We then perform a affine transformation using the parameters $\boldsymbol{\gamma}_l$ and $\boldsymbol{\beta}_l$. These parameters will be updated throughout training and can be viewed to model a variance and mean of the entire training set in each hidden layer. Thus we are enforcing the same mean and variance for all mini-batches.

Regularization Techniques

Machine learning and especially neural networks are prone to overfitting. Overfitting means that the model start memorizing training data instead of learning from it and generalizing that knowledge. This happens when the models explainatory capacity is to high compared to the problem at hand. The model will then tend to learn noise in the dataset believing it has to do with the underlying problem.

In order to avoid overfitting several regularization techniques can be applied. A regularization technique aims to penalize or make complex models more unlikely, i.e regularization aims to restrict the explainatory capacity of a machine learning and improve the models ability to generalize [5, p. 256-271].

Further, a well used regularization technique is *dropout* [25]. In dropout we assign a probability p to each neuron and for each propagation through the network we will temporary remove a neuron from the network with a probability of p, i.e we "drop" that neuron. This will make the neural network unable to use its full capacity and we can train exponentially many smaller neural networks at the same time [25].

In addition, when the training step is finished we can exploit the stochastic behavior in the neural network enforced by the dropout in order to get a more sophisticated prediction model. The dropout regularization can be seen as a a Bayesian approximation of the probabilistic Gaussian process [26]. Hence, we can view training with dropout as an approximation of complex probability distribution, rather than an approximation of a deterministic function. Therefore, we can use our neural network to sample several predictions for the same data point and use the sample mean as our final prediction and the sample variance as a uncertainty measure [26].

Furthermore, another common regularization technique is *weight regularization* [18, p. 226-233]. This technique adds a penalty term for large weights, with respect to some norm, to the loss function. The resulting problem becomes

$$\min_{\boldsymbol{W},\boldsymbol{\theta}} L(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) + \lambda ||\boldsymbol{W}||^2, \tag{3.34}$$

here $\boldsymbol{W}$ denotes the tensor containing all weight matrices between the layers in the neural network and λ the importance of the regularization. Penalizing the network based on the size of the weights will force "unnecessary" weights to zero since they are not contributing to minimzing the loss function. Encouraging small weights have also been linked to a better generalization.

However, it is also possible to perform regularization by changing the data. This can be done by adding noise to each data point, either just in the input phase or in every hidden layer as well [18, p. 236-239].

Weight Initialization

The biases in a neural network is often initialized to zero while the weights are drawn from some distribution. It is important to have some random element in the weight initialization in order to break symmetry. If all weights are initialized to the same value then the error associated with each weight in the gradient descent will be the same and the weights will be symmetric in all updates [27].

3.2.1.3 Ladder Network

A ladder network is a latent variable method making it possible to train a neural network in a semi-supervised setting [28].

In order to understand the idea behind the ladder network we first introduce the concept of an *autoencoder*. An autoencoder is a neural network trained in a unsupervised setting with the goal to copy its input to the output, i.e approximating the identity function. However, the number of neurons in the hidden layers of the autoencoder are strictly less than the number of neurons in the input and output layer. Consequently, the autoencoder first maps the input to a latent space of a lower dimensionality and then back to the original input space. This latent space can be viewed as a more compact representation of the data, often called *code*. Hence, the goal with an autoencoder is often to achieve a more compact representation of data and at the same time minimizing the loss of information. The part of the autoencoder mapping to the compact representation is often referred to as *encoder* while to part going from the compact representation back to the original space is the *decoder* [18, p. 499-506].

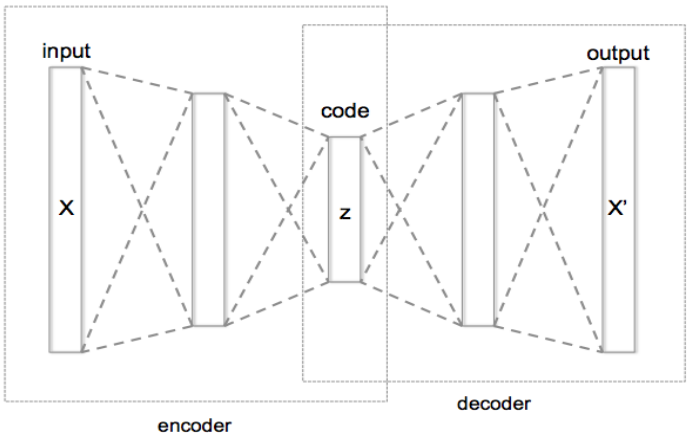


Figure 3.7: Illustration of the autoencoder structure. Courtesy of Chervinskii, figure published under CC BY-SA 4.0 [29].

We know briefly describe the overarching structure of the ladder network

1. Two encoders are created sharing the same weights and biases. One with a gaussian noise addition in each layer and one without any noise addition. The

encoder with noise is called the *corrupted encoder* and the one without is called the *clean encoder*.

2. We view the code produced by the encoders as predictions and given a data point $\boldsymbol{x}$ and we assign a supervised loss function, L_s, to the corrupted encoder
3. From the output layer of the corrupted encoder we add a decoder in order to form a autoencoder.
4. Between each layer in the corrupted encoder and the decoder we add skip connections in form of a denoising function between corresponding layers in the corrupted encoder and the decoder.
5. For each layer in the decoder we now define a unsupervised loss being the square loss between the reconstructed state in the decoder and the corresponding state in the clean encoder. For each layer we also assign a scaling factor u_l to corresponding unsupervised loss. The scaling factor u_l becomes the importance of the unsupervised loss in each layer. We refer to u_l as the *denoising cost*.

We can train the ladder network in a semi-supervised setting by evaluating the supervised loss on a annotated dataset S and the unsupervised loss functions on a unlabeled dataset U. For a detailed derivation of the ladder network we refer the reader to the original paper *Semi-Supervised Learning with Ladder Networks* by A.Rasmus et alt. [28]. The paper *Deconstructing the Ladder Network Architecture* by M.Pezeshki et alt. gives a thorough review of the elements in a ladder network [30].

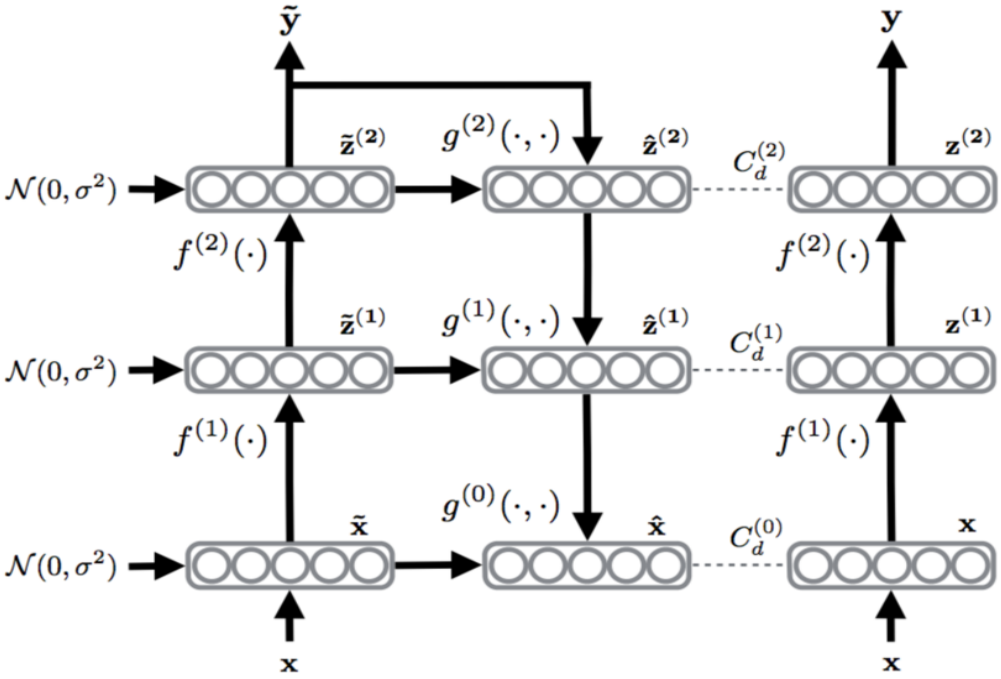


Figure 3.8: Illustration of a two layer ladder network. Note that we first compute a prediction $\hat{\boldsymbol{y}}$ and then tries to reconstruct the input $\boldsymbol{x}$ from $\hat{\boldsymbol{y}}$. The reconstruction loss is calculated by comparing the reconstruction to the internal states of the clean encoder.

3.2.2 Bag-of-Words

Bag-of-words is a concept from the field of *natural language processing* and is a simple method to get a vector representation of a text. Given a vocabulary of words, $\{word_i\}_{i=1}^{n}$, of length n and a text T. The corresponding vector, $\boldsymbol{x}$, of the text is created in the space $\boldsymbol{N}^n$ by letting each dimension, $i \in [n]$, representing a word, $word_i$, in the vocabulary and setting x_i to be the number of occurrences of $word_i$ in the text T [31].

The bag-of-word approach has its limitations since it does not consider the position of the words in the text when computing the vector representation. Instead it provides us with information regarding the overarching theme of the text.

When computing the bag-of-word vector for the DNA of a bacteria, viewing k-mers as words, we will refer to the resulting vector as *k-mer profile*.

3.2.3 t-distributed Stochastic Neighbor Embedding

The algorithm t-distributed Stochastic Neighbor Embedding (t-SNE) is used for non-linear dimensionality reduction and is well suited for visualization of high dimensional data [32].

The algorithm starts by computing a similarity between all points in the dataset. This similarity is based on the probability $p(\boldsymbol{x}_j|\boldsymbol{x}_i)$ of a point $\boldsymbol{x}_j$ being drawn from the dataset based on its probability density under a gaussian distribution centered at $\boldsymbol{x}_i$. The algorithm then computes a representation $\hat{\boldsymbol{x}}_i$ in a lower dimensional space of each point $\boldsymbol{x}_i$. The similarity between each point in the lower dimensional space is then computes as above, but under a Student t-distribution. The algorithm then tries to minimize to Kullback-Leibler divergence between the similarities in the original space and to lower dimensional space by adjusting the coordinates of the representations $\hat{\boldsymbol{x}}_i$ [32].

We can use t-SNE to get a representation of high dimensional in a space of lower dimension since the minimization of Kullback-Leibler divergence makes the t-SNE search for structures and clusters in the data.

However, the t-SNE do not preserve distance nor density [33]. Thus, the dimensions of the resulting space becomes hard to interpret due to the non-linear nature of the algorithm and t-SNE should only be used as a visualization tool.

3.3 Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction technique used in order to map high dimensional data into a feature space of lower dimension [5, p. 566-570].

In PCA the data is first normalized w.r.t to each feature. The covariance matrix, C, of the dataset is then computed and decomposed into eigenvectors and eigenvalues. These eigenvectors are referred to as *principal components* and forms a orthogonal basis of the space (or subspace) in which the data lies. Each eigenvalue will represent the fraction of the variance in the dataset explained by the corresponding principal component [34].

Given some threshold $\alpha < 1$ we can sort the eigenvalues and only keep the k first principal components satisfying

$$\alpha \leq \sum_{i=1}^{k} \lambda_i^2. \tag{3.35}$$

Mapping the original dataset onto these k principal components will yield a dimensionality reduction of the dataset keeping at least α of the variation in the dataset [34].

Calculating the principal components with corresponding eigenvalues can be done with the power method [35].

4
Methods and Material

In the following chapter we present the datasets and methods considered during the thesis work. The models considered will be based on a bag-of-word approach where the input will be a k-mer profile for each genome. We do this since the number of genomes available is much smaller than the actual size of a Salmonella genome. Due to this we cannot expect a model with a high resolution like a sequenced based model, *long short-term memory* or *gated recurrent unit*, to capture long term dependencies over the genomes and perform well. Calculating k-mer profiles will cause a loss in information regarding the context of each k-mer but given the size of the dataset this information loss might end up having a regularizing effect.

In addition, the machine learning models considered will be the following two

1. Feedforward Neural Network and
2. Ladder Network.

We will restrict our work to only consider models prediction the MIC-values for all antibiotics at the same time. Our justification for this is the present of missing labels in the labeled dataset which at first glance might seem as a contradiction. However, for some antibiotics the number of labeled datapoints is drastically smaller than the total number of datapoints and by considering all antibiotics on the same neural network we can possibly utilize correlations between antibiotics with a small count of labeled datapoints and antibiotics with a high count. The standard neural network does not provide a solution for missing labels so we present a custom softmax layer in section 4.6 solving this problem.

4.1 Datasets

Throughout this thesis-work we will consider one labeled dataset and one unlabeled dataset. Furthermore, we will restrict the work to only consider the bacteria *Salmonella enterica*.

4.1.1 Labeled Dataset

The labeled dataset we will use consists of 4964 *Salmonella enterica* samples and are collected from the dataset originally presented in the article *Using Machine Learning To Predict Antimicrobial MICs and Associated Genomic Features for Nontyphoidal Salmonella*[36]. This dataset consists of raw sequencing reads for each bacteria and we assemble them using the IDBA-UD assembler [37].

Further, the MIC-values in the dataset have a two-fold resolution, i.e all MIC-values and bounds are expressed as a $2^n, n \in \boldsymbol{Z}$. The labels in the dataset is somewhat inconsistent. For some samples the exact MIC-value is stated while for other samples we are instead given an upper-bound or a strict lower-bound. To handle this we set MIC-values stated with an upper-bound, $y \leq 2^n$, to be equal to the upper-bound $y = 2^n$. For the lower-bound $2^n < y$ we say that $y = 2^{n+1}$. These bounds are only at the thresholds presented in table 3.2 so by rounding them will not cause a miss classification with respect to the labels resistant or sensitive.

We now have 17 different MIC-values present in our dataset and we present the distribution of these for each antibiotic in table 4.1. The accession number for each bacteria along with its MIC-values are presented in supplementary file 2 of the article *Using Machine Learning To Predict Antimicrobial MICs and Associated Genomic Features for Nontyphoidal Salmonella* [36].

	0.007813	0.015625	0.03125	0.0625	0.125	00.25	0.5	1	2	4	8	16	32	64	128	256	512	Total
AMP	0	0	0	0	0	0	0	2950	479	27	3	2	2	1501	0	0	0	4964
AUG	0	0	0	0	0	0	0	3229	230	28	407	336	132	602	0	0	0	4964
AXO	0	0	0	0	0	4226	12	1	1	24	152	351	153	39	5	0	0	4964
AZI	0	0	0	0	0	0	0	12	517	1618	219	9	7	0	0	0	0	2382
CHL	0	0	0	0	0	0	0	0	42	1687	2988	87	8	152	0	0	0	4964
CIP	2154	1941	779	35	9	18	21	6	1	0	0	0	0	0	0	0	0	4964
COT	0	0	0	0	4543	328	35	2	1	3	52	0	0	0	0	0	0	4964
FIS	0	0	0	0	0	0	0	0	0	0	0	509	1306	1303	69	14	1508	4709
FOX	0	0	0	0	0	0	0	193	2460	1386	194	93	289	349	0	0	0	4964
GEN	0	0	0	0	0	1484	2413	355	31	10	67	171	433	0	0	0	0	4964
KAN	0	0	0	0	0	0	0	0	0	0	794	12	3	2	80	0	0	891
NAL	0	0	0	0	0	0	0	14	1247	3496	144	24	9	30	0	0	0	4964
STR	0	0	0	0	0	0	0	0	44	168	478	182	107	820	895	0	0	2694
TET	0	0	0	0	0	0	0	0	0	2207	27	22	152	2556	0	0	0	4964
TIO	0	0	0	0	0	9	1194	2895	149	8	135	574	0	0	0	0	0	4964

Table 4.1: The MIC-value distribution for each antibiotic in the dataset .

4.1.2 Unlabled Datasets

The unlabeled dataset is the GenBank *Salmonella enterica* dataset downloaded from *www.ncbi.nlm.nih.gov* the 2019.03.20 and consists of 17 674 assembled samples.

4.2 Predicting MIC-values

The output of our models will be non-parametric probability distributions over the present MIC-valuess in the labeled dataset

$$p(y_i = n|\mathbf{x}), \quad i \in A, \quad n \in \mathbf{Z} : n \in [-7, 9] \tag{4.1}$$

We choose to work probability distributions since they tends to contains more information than a single prediction.

Further, we only consider categorical distributions over the set $n \in \mathbf{Z} : n \in [-7, 9]$. This is due to the fact that the dataset only contains integer values within this interval and because working with 2 fold-dilutions is conventional within the field of clinical microbiology [38]. Hence, there is no need to predict a MIC-value not on this form in practise.

4.2.1 Accuracies

In this section we define the metrics used to evaluate the models. We choose to consider three different accuracies in order to capture the models ability to understand resistance on different levels of abstraction.

4.2.1.1 Exact Accuracy

The strictest metric will be what we refer to as "exact accuracy" and considers the models ability to predict the exact MIC-values. A prediction will be classified as correct if the model assigns most probability to the true MIC-value.

With this metric we want to evaluate how hard the problem of predicting exact MIC-values is and to get a hint of how complex the patterns between genotype and antibiotic resistance really are.

4.2.1.2 ±1 2-fold Dilution

The second metric we consider is the model's ability to predict the correct MIC-values within a ±1 2-fold dilution step. A predicted MIC-value $\hat{y} = 2^n$ is classified correctly if the true y satisfies $y \in [2^{n-1}, 2^{n+1}]$.

We consider this accuracy since it is used to evaluate the model in the article *Using Machine Learning To Predict Antimicrobial MICs and Associated Genomic Features for Nontyphoidal Salmonella* [36].

4.2.1.3 3-class accuracy

In many cases one is more interest in knowing whether or not a bacteria is resistant to an antibiotic rather than knowing the exact MIC-value. Thus, we will also consider a metric where we translate the output of a model to the labels sensitive, intermediate and resistant based on the clinical guidelines presented in table 3.2 and compare whether or not the label is the same for the true MIC-value.

4.2.2 U.S Food and Drug Administration Error-rates

The U.S Food and Drug Administration recommends evaluating a automated prediction model using three different error-rates [38]. These are based on the confusion matrix in table 4.2 and are defined as follows

1. Very-Major error-rate
$$E_{vmj} = \frac{\text{FN}}{\text{FN + TP}}, \tag{4.2}$$
2. Major error-rate
$$E_{maj} = \frac{\text{FP}}{\text{FP + TN}}, \tag{4.3}$$
3. Minor error-rate
$$E_{min} = \frac{FIN + IN + IP + FIP}{\text{FN + TN + FP + TP + FIN + IN + IP + FIP}}. \tag{4.4}$$

		Predicted Label		
		Sensitive	Intermediate	Resistance
True Label	Sensitive	TN	FIN	FP
	Intermediate	IN	TI	IP
	Resistance	FN	FIP	TP

Table 4.2: Confusion matrix used for calculating the error-rates defined above.

Given a double sided confidence interval of 95%, The U.S Food and Drug Administration standards indicates a lower bound of the interval below 1.5% and a upper bound the interval of at most 7.5% for the very-major error-rate . The major error-rate should be below 3% [38].

4.3 Baseline

In order to evaluate and discuss around the models we have to set them in relation with something else and study the distribution of the data. For example, a very complex model might have an accuracy of say 95%, but is this a good accuracy? Maybe 95% of the targets in the dataset are the same? Hence, just considering the magnitude of an accuracy is not interesting and in order to interpret the accuracy and we need further information.

To create a baseline, which we consider our model in relation to, we will first investigate the distribution of the MIC-values in the training set, see table 4.1. A baseline model will then be created, that for each antibiotic predicts the most occuring MIC-value.

To conclude that a machine learning model as actually learned to correlate variance in the input space with variance in the output space we require that the accuracy is strictly above the baseline. This since a accuracy strictly above the baseline model shows the machine learning model has made a active decision based on the input.

4.4 Pre-Processing

The given data contains both raw reads and assembled contigs. We start by assembling the raw reads using the idba assembler [37]. We then for, all contigs, calculate the occurence of overlapping k-mers of size 10 using the kmer-counting program KMC [39]. The kmer count is then used to produce a bag-of-word, or kmer profile, vector for each sample, see algorithm 1.

Algorithm 1: Computing the k-mer profiles

```
kmer_dictionary
Dataset = zeros(nbr_samples, 4^k)
for sample in samples do
    present_kmers, kmer_counts = KMC(sample)
    initialize kmer_profile
    for kmer in present_kmers do
        i = kmer_dictionary.get_key(kmer)
        kmer_profile[i] = kmer_counts[kmer]
    end
    Dataset[sample, :] = kmer_profile
end
```

Due to the exponential space-complexity, $O(4^k)$, of the corresponding vector space we perform dimensionality reduction through principal component analysis (PCA), see section 3.3.
First we randomly split the labeled dataset into three parts, training-, validation- and test-set. We learn the PCA transform using only the labeled training-set with the explainability threshold set to $\alpha = 0.995$. The unlabeled dataset where not included in learning the PCA transform due to computational restrictions. In algorithm 2 we present the psuedocode for the dimensionality reduction.

Algorithm 2: Dimensionality reduction

D_s
D_u
random_seed
$\alpha = 0.995$
D_{train}, $D_{validation}$, D_{test} = random_split(D_s, random_seed)
μ_{train} = feature_mean(D_{train})
σ^2_{train} = feature_var(D_{train})
$X_{train} = \frac{D_{train} - \mu_{train}}{\sigma^2_{train}}$
$X_{validation} = \frac{D_{train} - \mu_{train}}{\sigma^2_{train}}$
$X_{test} = \frac{D_{train} - \mu_{train}}{\sigma^2_{train}}$
transform = learn_pca_transform(X_{train}, α)
D_{train} = transform(D_{train})
$D_{validation}$ = transform($D_{validation}$)
D_{test} = transform(D_{test})
D_{non} = transform(D_u)

4.5 Oversampling the Training Data

As stated in section 4.1.1, the distribution of the MIC-values are highly skewed for certain antibiotics. This might obstruct the learning process since predicting the

mode becomes a local minima. In order to avoid this we would like the training appear uniformly distributed.

4.5.1 Bootrapping

One way to make the data uniformly distributed is to apply bootstrapping in order to even the count for all MIC-values. This is done by first computing the count of the most occurring MIC-value for each antibiotic. We then for each antibiotic computes the subset of samples corresponding the the MIC-values not occurring as much as the most common MIC-value. From these sets we start randomly drawing new samples to the training dataset until the corresponding MIC-value occurs in the same amount as the most occurring MIC-value for the corresponding antibiotics.

However, naively sampling from these subsets will not balance the distribution of MIC-values since the dataset contains missing data. Thus, when sampling for a specific antibiotics and MIC-value we set all the other MIC-values for the auxiliary antibiotics to be missing.

Moreover, oversampling might amplify the effects of outliers if not done correctly. Consider a MIC-value that only occur once and which is a product of some error in the measurement equipment. Oversampling would increase the occurrence of this MIC-value and thus amplify the error. To prevent this, we require the subset sampled from to contain at least 10 different samples.

4.5.2 Synthetic Minority Over-sampling Technique

In order to create a more sophisticated oversampling procedure than bootstrapping we will use mathematical reasoning.

We assume that the function g which we will try to approximate using machine learning is locally constant in the feature space. This is reasonable since the DNA is a discrete structure and mutations are binary events. A genome can thus not contain a fraction of a mutation and only vectors having only integral elements corresponds to a possible true bacteria.

Thus, given a dataset D we assume that for any $\boldsymbol{x}_1 \in D$ with $g(\boldsymbol{x}_1) = \boldsymbol{y}$ the following is true

$$\exists \epsilon > 0 : g(\boldsymbol{x}_{new}) = \boldsymbol{y}, \quad \forall \boldsymbol{x}_{new} \in \{\boldsymbol{x} \in \boldsymbol{X} : \quad ||\boldsymbol{x} - \boldsymbol{x}_1|| < \epsilon\} \tag{4.5}$$

Hence, we can sample new data points to our dataset from the set defined in equation 4.5. Our hypothesis is that this will enhance topological structures for the underrepresented MIC-classes in our dataset.

In practise, this done by applying *Synthetic Minority Over-sampling Techniques* (SMOTE) [40]. For each antibiotic we compute the subset of samples corresponding to each MIC-value in the training set

$$D_j^i = \{\boldsymbol{x} \in D_{training} : g(\boldsymbol{x})_i = j\}, \quad i \in A, \quad j \in \mathbf{Z} : j \in [-7, 9] \tag{4.6}$$

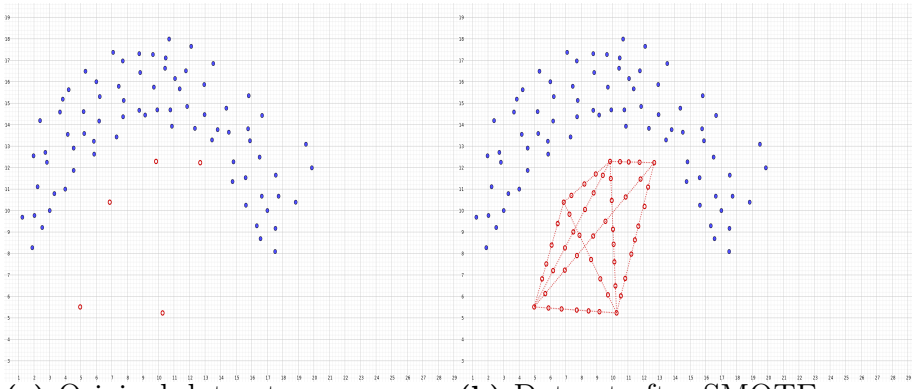
We then draw a random sample $\boldsymbol{x}_1$ from a set D_j^i corresponding to a underrepresented MIC-value, j, of an antibiotic i. For $\boldsymbol{x}_1$ we then compute the set K containing

the k-nearest neigbors of $\boldsymbol{x}_1$ in D_j^i with respect to the euclidean norm. From the set K we randomly draw another point $\boldsymbol{x}_2 \in K$ and construct an new sample by

$$\boldsymbol{x}_{new} = \boldsymbol{x}_1 + (1 - r)(\boldsymbol{x}_1 - \boldsymbol{x}_2), \quad r \sim U(0, 1). \tag{4.7}$$

We set $y_{new,i} = j$ and treat the MIC-value for all other antibiotics as missing in order to ensure a uniform distribution in the end. We loop this process until the MIC-values are uniformly distributed for all antibiotics. A toy example using SMOTE is presented in figure 4.1.

To not amplify the effects of outliers we require a MIC-value to occur at least 10 times in order be enhanced by this process.



(a) Original dataset **(b)** Dataset after SMOTE

Figure 4.1: Illustration of how the dataset changes when SMOTE is applied. The dashed lines are the ones from which we sample new datapoints.

4.6 Training and Evaluation

We will implement the standard feedforward neural network along with its extension ladder network, which we present in section 3.2.1.3.

Further, we set the activation function to be *Rectified Linear Unit* (ReLU)

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{4.8}$$

The activation in the output layer will be the *softmax* activation function

$$h_i = \sum_j w_{j,i} z_i + \theta_i, \tag{4.9}$$

$$f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})_i = \frac{e^{h_i}}{\sum_{j=1}^{dim(\boldsymbol{Y})} e^{h_j}}. \tag{4.10}$$

where h_i is the sum of the input signals to the output node i as defined in section 3.2.1. The softmax activation will squeeze the output to $f_{\boldsymbol{w},\boldsymbol{\theta}}(\mathbf{x})_i \in [0, 1]$ such that $\sum_{i=1} f_{\boldsymbol{w},\boldsymbol{\theta}}(\mathbf{x})_i = 1$, i.e we can interpret the output as a probability distribution.
The ouput layer will be a concatenation of 15 softmax layers. This since we are

considering the MIC-values for 15 different antibiotics and by using one neural network to predict them all we might be able to capture correlation between different antibiotics.

Further, the labeled dataset is not complete. For many samples there are missing MIC-values. To handle this we will ignore the gradient from the corresponding softmax layer whenever a missing MIC-value is encountered.

As a loss-function we choose the cross-entropy between the output distribution and the true distribution

$$L(f_{w,\theta}(\mathbf{x}), \boldsymbol{y}) = -\frac{1}{|S|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in S} \boldsymbol{y}^T \log(f_{\boldsymbol{w},\boldsymbol{\theta}}(\boldsymbol{x})), \tag{4.11}$$

where $\boldsymbol{y}$ is one-hot encoding of the true MIC-value. Since our labeled dataset is quite small we restrict our neural network to have at 2750 neurons and a maximum of two hidden layers (the number of maximum neurons is roughly the number of dimensions after the dimensionality reduction).

Moreover, we will train our model using the regularization techniques presented in the chapter 3. The norm for the weight regularization will be the *Manhattan-norm* $||\boldsymbol{w}||_{L_1}$. Noise will be added to the input layer and the hidden layers and we choose to use gaussian noise $N(0, \sigma^2)$. Dropout will be implemented in all layers of the neural network. For the prediction step we will sample 100 predictions for each input and output the average over these predictions as our final prediction.

We will also test whether training the feedforward neural network as a ladder network can improve the accuracy of the model. The importance of the unsupervised learning will be decided by the denoising cost for each layer in the network.

To find a good set of hyperparameters we will use Bayesian optimization and especially the *Tree-structured Parzen Estimator* algorithm w.r.t to the exact accuracy on the validation set [41].

When a good set of hyperparameters have been found we evaluate the oversampling methods compared to using no oversampling method using a nested 5-cross validation, illustrated in figure 4.2.

We will refer to a network trained with no oversampling method applied as *no-sampling model*, a model trained using bootstrap on the training set will be referred to as *bootstrap model* and when SMOTE is used on the training set we refer to the resulting neural network as *SMOTE model*. We will pick the best performing model of these three and refer to it as *DeepMIC*. The DeepMIC model will then be further analyzed with respect to the 2-fold accuracy, 3-class accuracy and the error-rates presented in section 4.2.

Furthermore, the DeepMIC model will also be compared to the model presented in *Using Machine Learning To Predict Antimicrobial MICs and Associated Genomic Features for Nontyphoidal Salmonella* [36] we refer to their model as *XGBoost* and to 1928 Diagnostics pipeline which we refer to as *1928D*.

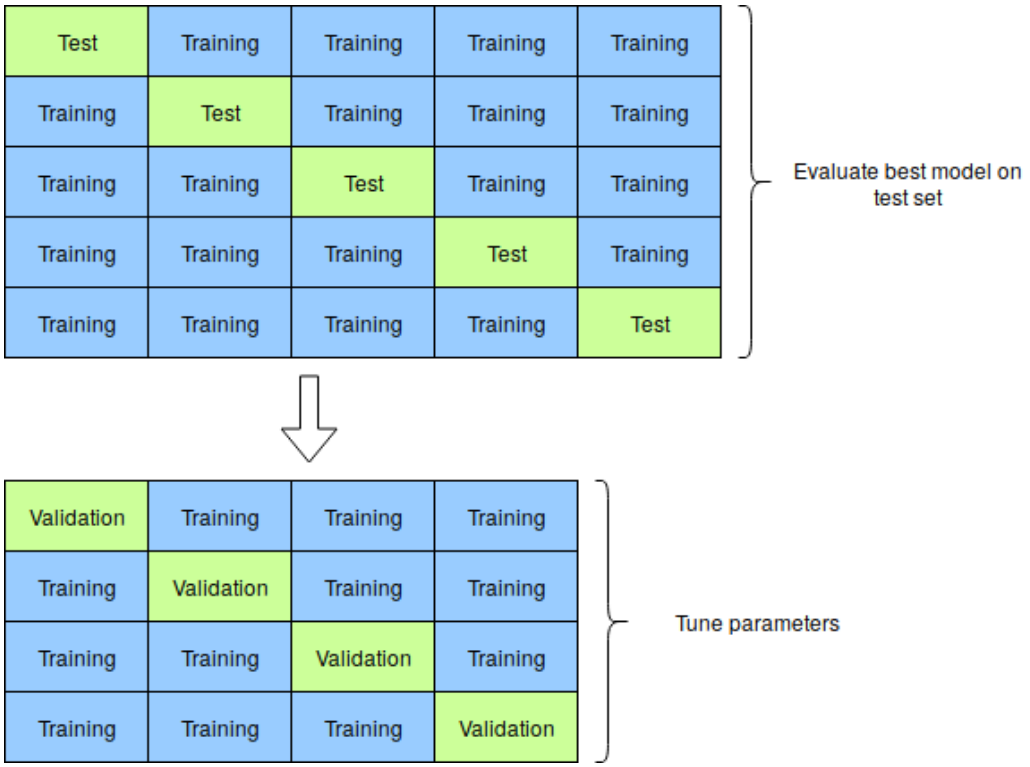


Figure 4.2: Illustration of the nested 5-fold cross validation.

4.7 1928 Diagnostics pipeline

The algorithms 1928 Diagnostics uses today are based on curated databases. This makes direct comparison with our prediction model impossible since our model predicts a MIC-value while 1928 Diagnostics model outputs genes, which can be linked to antibiotic resistance, found in the bacteria genome. In order to do some comparison we run 1928 Diagnostics model for the antibiotics ampillicin, streptomycin and tetracycline. For these antibiotics we say that 1928 Diagnostics model outputs the label resistant if one of the genes presented in appendix C.1 where found.

4.8 Miscellaneous

The computations were performed on resources at Chalmers Centre for Computational Science and Engineering (C3SE) provided by the Swedish National Infrastructure for Computing (SNIC).

The programming language used is Python version 3.6.7 and in table 4.3 follows the packages used and where in the process there where used.

Python Package	Where?
Scikit-Learn [42]	Pre-processing, Visualization
Matplotlib[43]	Visualization
Tensorflow [44]	Neural Network Implementation
OLCTools	KMC Python Support
Numpy [45]	Data representation
Pandas [45]	Visualization

Table 4.3: Python packages used and where in the process we used respective package

5

Results

In this section we present the result of our models. We start by presenting the result of the pre-processing step, followed by a comparison of models trained on the different oversampling techniques. We then pick the best performing model as our final model and compare it to the XGBoost model and 1928 Diagnostics platform. This is followed by a illustration of how different hyperparameters affects the model. We end this chapter by visualizing how our neural network is able to detect antibiotic resistance.

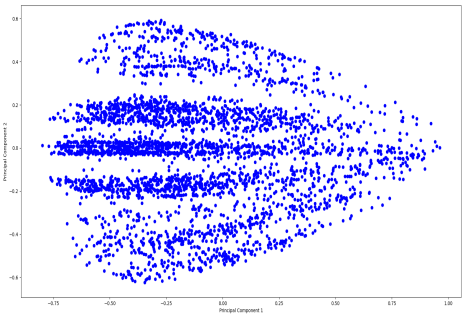
5.1 Pre-Processing and Visualization of the dataset

Depending on the random split, the resulting feature spaces after PCA dimensionality reduction ended up have a dimensionality within the interval $[2617 - 2701]$ with a median at 2621 dimensions.

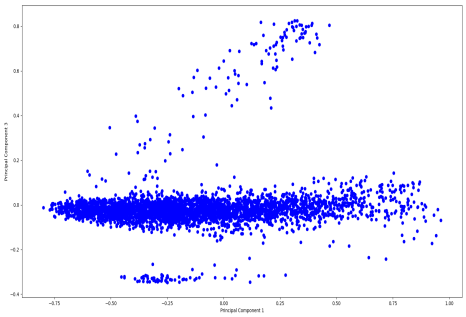
In order to explore the dataset we plot the three principal components explaining most of variance against each other in figure 5.1. All samples seems grouped together and we cannot distinguish any clusters. For these three principal components the explainability on average is

- principal component 1: 0.30,
- principal component 2: 0.05,
- principal component 3: 0.04.

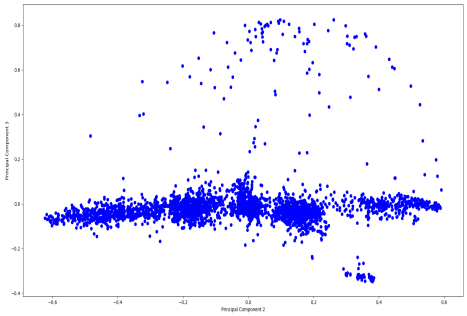
Hence, the cumulative explainability of these components is 0.39 meaning that 0.61 of the variance is explained by other components in the feature space with each of these components having an explainability of at most 0.04. Thus, the variance in the data is distributed over many components in the feature space.



(a) Principal component 1 and 2



(b) Principal component 1 and 3



(c) Principal component 2 and 3

Figure 5.1: The three principal components explaining the most variance in the data. We cannot distinguish any clear clusters.

Nevertheless, when applying t-SNE to the space of principal components clusters emerges, see figure 5.2. In figure 5.3 we have colored each sample w.r.t to tetracycline resistance and we can clearly see structures form containing resistant or sensitive samples. The presents of clusters of resistant samples points towards that our pre-processing step uncovers patterns linked to antibiotic resistance. For figure 5.2 colored w.r.t each antibiotic see appendix A.1.

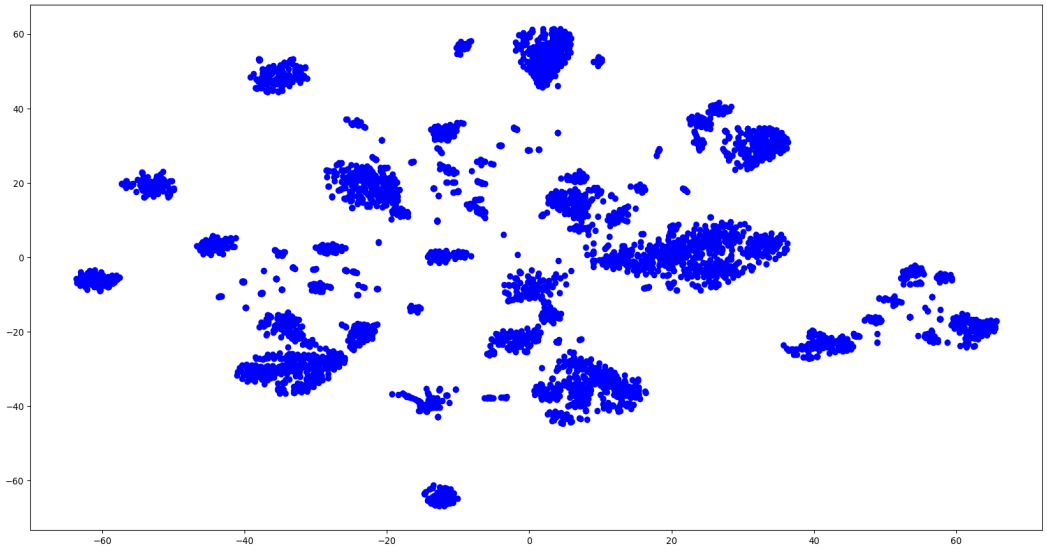


Figure 5.2: Annotated data visualized using the manifold learning-algorithm t-SNE. The perplexity was set to 50.

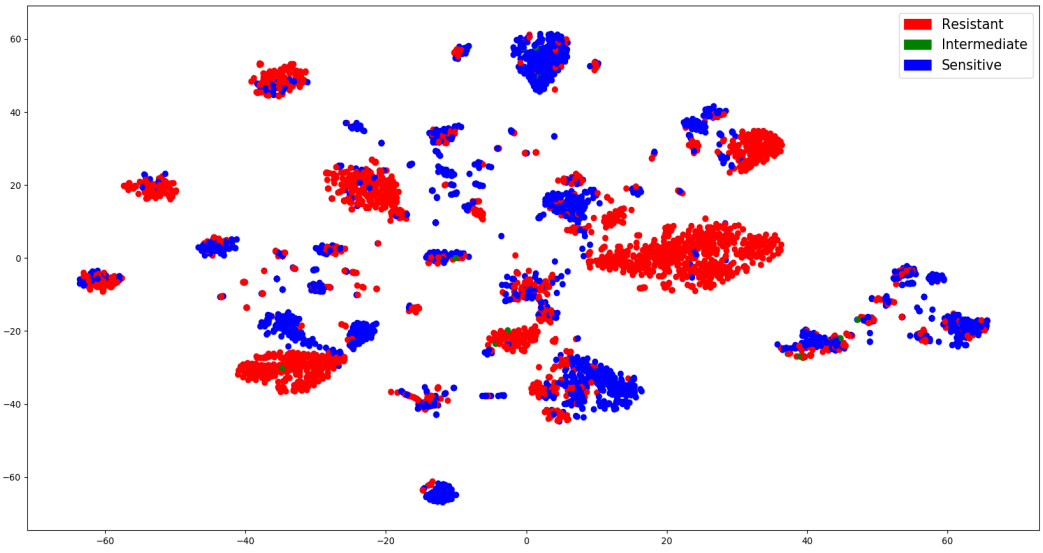


Figure 5.3: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect to tetracycline resistance. The perplexity was set to 50. The k-mer profiles from resistant bacteria seems to form cluster. Hence there is a signal to learn in the dataset.

5.2 Model Evaluation

The hyperparameters found through the Parzen Tree estimator are presented in table 5.1. The optimal denosing costs where found to be 0, i.e the semi-supervised learning where found to not improve the accuracy. In section 5.3 we present the result for the semi-supervised learning in more detail.
Moreover, the weight regularization where found to be redundant in the present of dropout and noise addition.

Hidden Layers	1
Neurons Hidden Layer	2250
Dropout Input	0.05
Dropout Hidden	0.3
Dropout Output	0.1
Initial Learning Rate	0.0005
Batch Size	500
Denoising Cost Input	0
Denoising Cost Hidden	0
Weight Regularization	0
Noise Addition σ^2	0.03

Table 5.1: The optimal parameters found by the Parzen Tree Estimator

In table 5.2 we present the mean accuracy on the test set for each oversampling technique along with a 95% confidence interval .We observe that there is no significant difference between the different oversampling methods and training without oversampling. However, from figure 5.4 we observe that the neural network trained on data over-sampled with SMOTE converges faster. Therefore, we will choose that approach to be our final model and we will from now on refer to it as *DeepMIC*. The average accuracy for prediction of exact MIC-values is 0.78 for the DeepMIC model.
Additionally, the oversampling boosted the training set from an average of 3176 samples to an average of 99 916 samples.

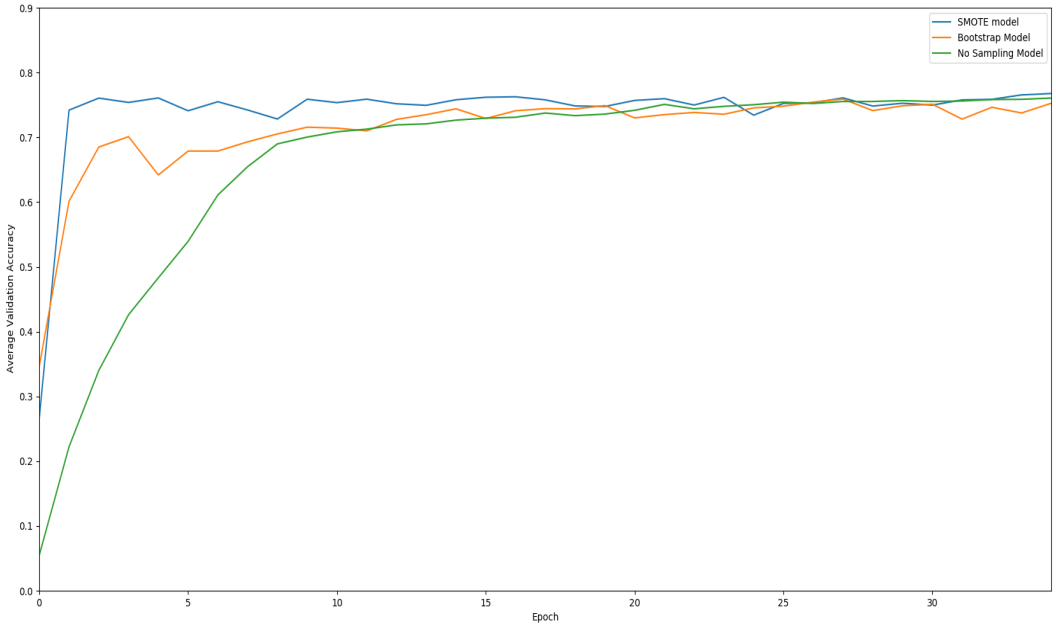


Figure 5.4: Accuracy averaged over antibiotics and runs for the SMOTE, bootstrap and no sampling model, ploted against epochs.

	No Sampling model		Bootstrap model		SMOTE model	
Antibiotics	Mean Accuracy	95% Confidence	Mean Accuracy	95% Confidence	Mean Accuracy	95 % Confidence
AMP	0.85	[0.84–0.86]	0.85	[0.84–0.86]	0.84	[0.83–0.85]
AUG	0.85	[0.85–0.85]	0.85	[0.84–0.86]	0.85	[0.84-0.86]
AXO	0.93	[0.93–0.93]	0.93	[0.93–0.93]	0.93	[0.93-0.93]
AZI	0.70	[0.68–0.72]	0.71	[0.70–0.72]	0.69	[0.67-0.71]
CHL	0.76	[0.75–0.77]	0.76	[0.76–0.77]	0.76	[0.75-0.77]
CIP	0.77	[0.76–0.78]	0.77	[0.76–0.78]	0.79	[0.78-0.80]
COT	0.91	[0.90–0.92]	0.93	[0.92–0.94]	0.92	[0.91–0.93]
FIS	0.66	[0.66–0.66]	0.66	[0.66–0.66]	0.66	[0.65-0.67]
FOX	0.71	[0.70–0.72]	0.71	[0.70–0.72]	0.71	[0.70-0.72]
GEN	0.61	[0.60–0.62]	0.60	[0.59–0.61]	0.61	[0.60–0.62]
KAN	0.92	[0.92–0.92]	0.91	[0.90–0.92]	0.92	[0.91-0.93]
NAL	0.74	[0.73–0.75]	0.76	[0.75–0.77]	0.76	[0.75-0.77]
STR	0.62	[0.60–0.64]	0.59	[0.57–0.61]	0.61	[0.58–0.64]
TET	0.93	[0.93–0.93]	0.93	[0.92–0.94]	0.93	[0.93-0.93]
TIO	0.74	[0.73–0.75]	0.74	[0.73–0.75]	0.73	[0.71–0.75]

Table 5.2: The result for different oversampling methods in the nested 5-cross fold validation with a 95% confidence interval.

Moreover, in table 5.3 we present the error-rates defined by the FDA. We remind the reader that FDA recommends a very major error-rate with an lower confidence bound under 1.5% and a major error-rate below 3% in order for the model to be suitable for use in a clinical environment. The DeepMIC model struggles with the very major error-rates and is for no antibiotic significantly below the FDA recommendation. For the major error-rate, the DeepMIC are below the recommendation for 10 of the 15 antibiotics.

	Very Major error-rate		Major error-rate		Minor error-rate	
Antiobiotics	Mean error-rate	95% Confidence	Mean error-rate	95% Confidence	Mean error-rate	95% Confidence
AMP	0.116	[0.099-0.132]	0.036	[0.023-0.048]	0	[0.000-0.000]
AUG	0.108	[0.087-0.129]	0.006	[0.003-0.009]	0.011	[0.006-0.016]
AXO	0.152	[0.135-0.168]	0.006	[0.005-0.007]	0	[0.000-0.000]
AZI	-	-	0	[0.000-0.000]	0	[0.000-0.000]
CHL	0.139	[0.127-0.152]	0.004	[0.002-0.006]	0.004	[0.002-0.006]
CIP	-	-	0	[0.000-0.000]	0	[0.000-0.000]
COT	0.805	[0.704-0.905]	0.001	[0.000-0.003]	0	[0.000-0.000]
FIS	0.058	[0.043-0.073]	0.031	[0.017-0.044]	0	[0.000-0.000]
FOX	0.121	[0.106-0.134]	0.005	[0.004-0.006]	0.003	[0.001-0.005]
GEN	0.122	[0.101-0.134]	0.013	[0.011-0.015]	0.003	[0.002-0.004]
KAN	0	[0.000-0.000]	-	-	0.033	[0.030-0.036]
NAL	0.867	[0.791-0.943]	0.002	[0.000-0.004]	0	[0.000-0.000]
STR	0.046	[0.038-0.054]	0.153	[0.099-0.206]	0	[0.000-0.000]
TET	0.042	[0.036-0.047]	0.043	[0.034-0.051]	0.001	[0.000-0.002]
TIO	0.099	[0.088-0.109]	0.006	[0.005-0.007]	0	[0.000-0.000]

Table 5.3: Error-rates for the DeepMIC model. For azithromycin and ciprofloxacin there were no resistant sample in the dataset. For kanamycin there was no samples within the intermediate region.

5.2.1 Comparisons with other Prediction Models

In table 5.6 we present the comparison between DeepMIC, the baseline method used and the XGBoost model. For 12 of 15 antibiotics we observe an accuracy significantly larger than the baseline which means that given an input, the DeepMIC model makes an active decision regarding these 12 antibiotics. We summarize the result presented in table 5.6 in table 5.4.

	Baseline	XGBoost model	DeepMIC
Average Accuracy	0.59	0.59	0.78

Table 5.4: Weighted average accuracy over all antibiotics for prediction of exact MIC-values.

Further, the DeepMIC model outperforms the XGBoost model on 14 of the 15 antibiotics considered. Only for the antibiotic Kanamycin have the two models an accuracy within the same confidence interval. If we regard the baselines accuracy for Kanamycin we observe an average of 0.89 which is comparable to the accuracy of the two machine learning models. Hence, predicting a MIC-value for Kanamycin in the given dataset is a trivial task.

Furthermore, in table 5.7 we present a comparison for the 3-class accuracy between the baseline model and the DeepMIC model. From the Baseline method we conclude that there is a significant skewness in the dataset towards a certain label for the antibiotics azithromycin, chlotamphenicol, ciprofloaxin, trimethoptim, kanamycin and nalidixix acid. The 3-class accuracy for these antibiotics are therefore not so intresting to study. We summarize table 5.7 in table 5.5

	Baseline	DeepMIC
Average Accuracy	0.80	0.97

Table 5.5: Summary of the result in the 3-class comparison

	Baseline		XGBoost model		DeepMIC	
Antibiotics	Mean Accuracy	95% Confidence	Mean Accuracy	95% Confidence	Mean Accuracy	95% Confidence
AMP	0.58	[0.57-0.59]	0.33	[0.29-0.36]	0.84	[0.83–0.85]
AUG	0.65	[0.64-0.66]	0.48	[0.45-0.51]	0.85	[0.84-0.86]
AXO	0.85	[0.84-0.86]	0.80	[0.78-0.83]	0.93	[0.93-0.93]
AZI	0.68	[0.66-0.70]	0.58	[0.55-0.62]	0.69	[0.67-0.71]
CHL	0.59	[0.58-0.60]	0.72	[0.70-0.73]	0.76	[0.75-0.77]
CIP	0.41	[0.40-0.42]	0.42	[0.41-0.43]	0.79	[0.78-0.80]
COT	0.92	[0.92-0.92]	0.87	[0.86-0.88]	0.92	[0.91–0.93]
FIS	0.33	[0.32-0.34]	0.57	[0.54-0.59]	0.66	[0.65-0.67]
FOX	0.50	[0.49-0.51]	0.58	[0.56-0.59]	0.71	[0.70-0.72]
GEN	0.48	[0.47-0.49]	0.46	[0.45-0.48]	0.61	[0.60–0.62]
KAN	0.89	[0.87-0.91]	0.91	[0.88-0.94]	0.92	[0.91-0.93]
NAL	0.70	[0.69-0.71]	0.62	[0.60-0.64]	0.76	[0.75-0.77]
STR	0.34	[0.33-0.35]	0.51	[0.49-0.52]	0.61	[0.58–0.64]
TET	0.51	[0.50-0.51]	0.47	[0.45-0.49]	0.93	[0.93-0.93]
TIO	0.59	[0.58-0.60]	0.73	[0.72-0.74]	0.73	[0.71–0.75]

Table 5.6: The DeepMIC model compared to the Baseline and XGBoost model w.r.t to exact accuracy. Accuracies for the XGBoost model taken from supplementary file 2 in the article *Using Machine Learning To Predict Antimicrobial MICs and Associated Genomic Features for Nontyphoidal Salmonella.* [36]

	Baseline		DeepMIC	
Antibiotics	Mean Accuracy	95% Confidence	Mean Accuracy	95% Confidence
AMP	0.69	[0.68-0.70]	0.94	[0.93-0.95]
AUG	0.78	[0.77-0.79]	0.94	[0.93-0.95]
AXO	0.85	[0.84-0.86]	0.98	[0.98-0.98]
AZI	1	[1.00-1.00]	1	[1.00-1.00]
CHL	0.95	[0.94-0.96]	0.97	[0.97-0.97]
CIP	1.0	[1.00-1.00]	1	[1.00-1.00]
COT	0.99	[0.99-0.99]	0.99	[0.99-0.99]
FIS	0.34	[0.32-0.34]	0.96	[0.96-0.96]
FOX	0.85	[0.84-0.86]	0.96	[0.96-0.96]
GEN	0.87	[0.87-0.87]	0.96	[0.96-0.96]
KAN	0.91	[0.88-0.93]	0.94	[0.92-0.96]
NAL	0.99	[0.99-0.99]	0.99	[0.99-0.99]
STR	0.63	[0.62-0.64]	0.92	[0.90-0.94]
TET	0.55	[0.54-0.56]	0.95	[0.95-0.95]
TIO	0.86	[0.85-0.87]	0.98	[0.98-0.98]

Table 5.7: 3-class accuracy for the Baseline and DeepMIC model. The XGBoost model was not evaluated w.r.t to this accuracy in the original article.

In table 5.8 we present the error-rates for the prediction model built around 1928 Diagnostics pipeline for the antibiotics ampillicin, sulfidoxazole and tetracycline.

	Very Major error-rate	Major error-rate	True Positives	True Negatives
AMP	0.077	0.007	1501	3442
STR	0.088	0.080	1713	973
TET	0.106	0.010	2722	2197

Table 5.8: The output from the prediction model built around 1928 Diagnostics pipeline.

5.3 Importance of Denoising Cost

The average accuracy over the antibiotics is ploted as a function of the denoising costs in figures 5.5 5.6. We can clearly see that an increased denoising cost corresponds to a decreased average accuracy.

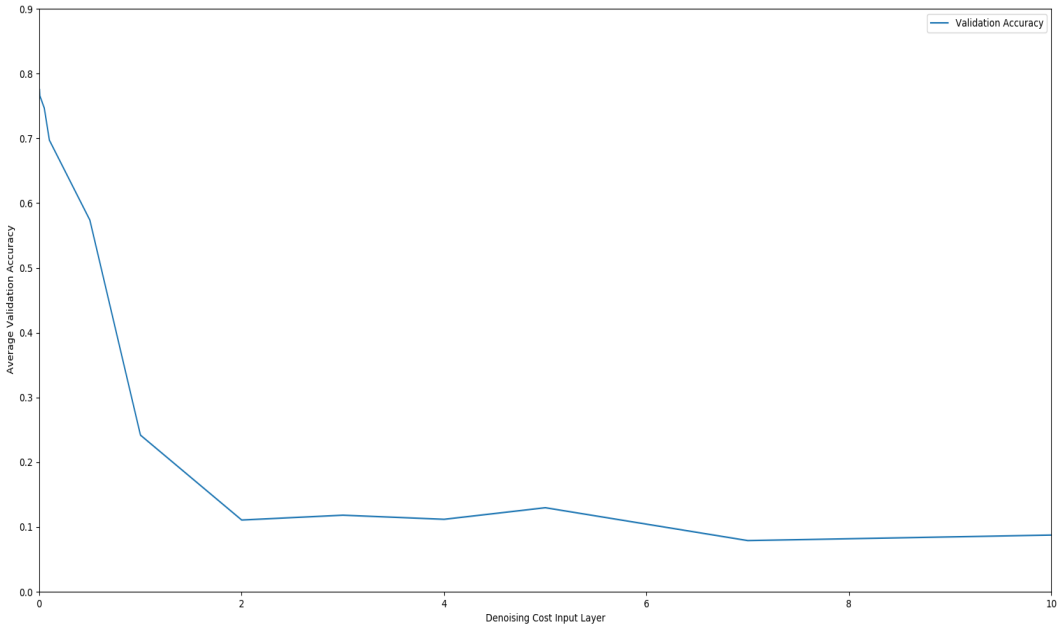


Figure 5.5: Accuracy averaged over antibiotics ploted against the denoising cost in the input layer for the ladder network. All other parameters where as in table 5.1.

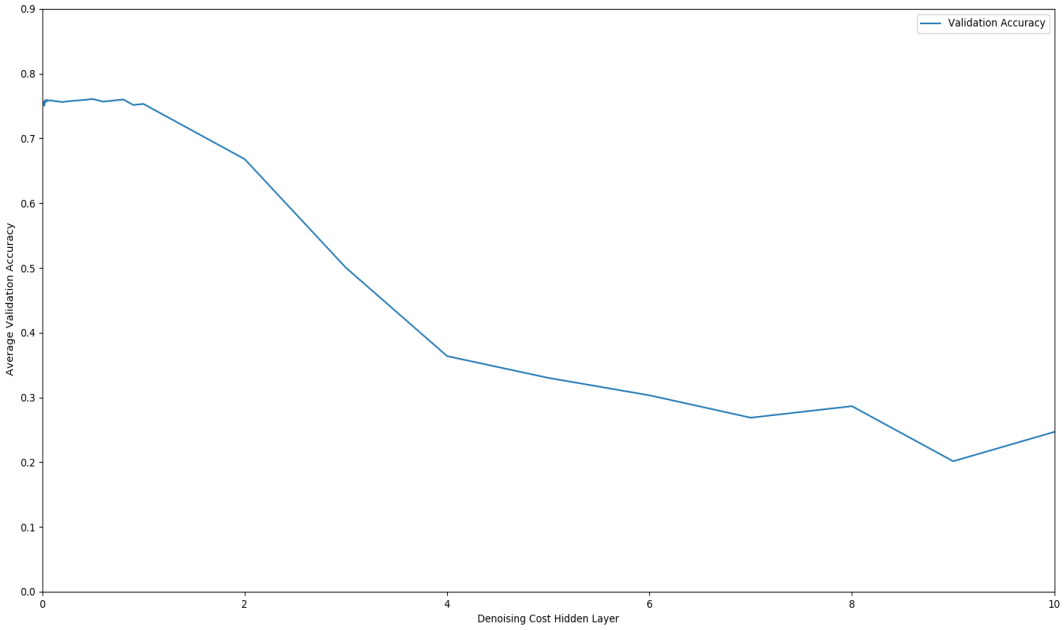
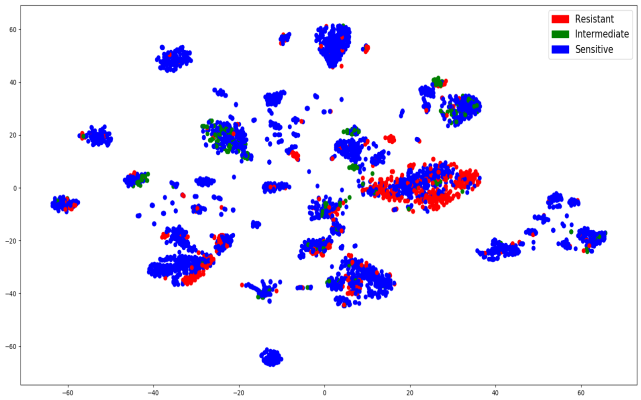


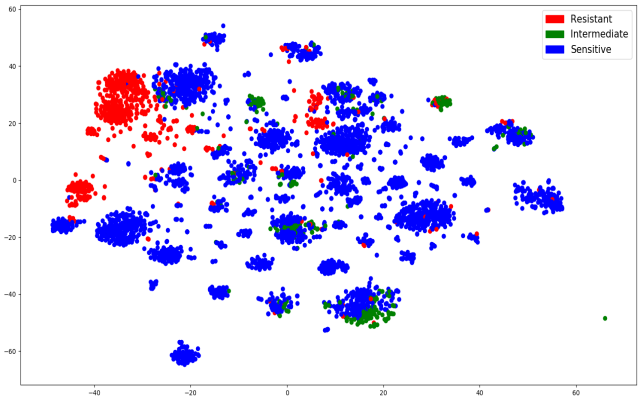
Figure 5.6: Accuracy averaged over antibiotics ploted against the denoising cost in the hidden layer for the ladder network. All other parameters where as in table 5.1.

5.4 Peeking into the Mind of an AI

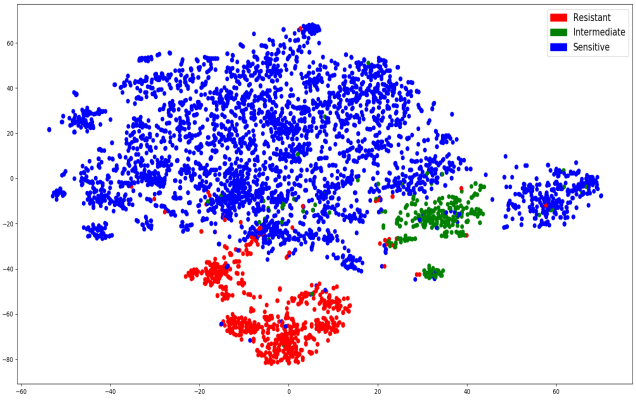
The DeepMIC model is a chain of non-linear mappings between very high-dimension spaces. To visualize this chain of mappings, we propagate the annotated dataset through the network, performing a 2-dimensional embedding using t-SNE in each layer. In figure 5.7 we illustrate the result, coloring each sample with respect to resistance towards amoxicillin. We can clearly see how samples with the same label are grouped together as the dataset traverse through the neural network. For figures colored w.r.t to other antibiotics see appendix A.1.



(a) Input layer



(b) Hidden layer



(c) Output layer before softmax

Figure 5.7: Visualization of each layer in the DeepMIC neural network using t-SNE. Colored w.r.t amoxicillin resistance. The perplexity was set to 50. This shows that bacteria having the same label activates the same neurons in the neural network.

6
Discussion

In this chapter we start by discussing around the DeepMIC model and then proceed to talk about general limitations for machine learning models within the field of antibiotic resistance. We end this chapter by presenting possible future work and how this can contribute to the field.

6.1 Model Performance

In this thesis we present a neural network based machine learning model for prediction of minimum inhibitory concentration for 15 different antibiotics with an overall average accuracy of 0.78 for prediction of exact MIC-value and 0.97 when the MIC-values are translated to labels. By comparing to a baseline model we can ensure that our network must do an active decision for 12 of the 15 antibiotics in order to achieve their respective accuracy.

The model outperforms the already existing extreme gradient boosted model presented by Marcus Nguyen et al. [36]. The biggest difference occurs for the antibiotics ampillicin(AMP), amoxicillin (AUG) and tetracycline (TET) for which our model has a mean accuracy of 0.84, 0.85 respectively 0.93 compared to 0.33, 0.48 and 0.47 for the gradient boosted model.

The result for ampillicin and tetracycline is somewhat expected. For both these antibiotics most of the data is distributed over two MIC-values, one corresponding to sensitive and one to resistance. Thus, for these two antibiotics the problem of predicting MIC-values becomes somewhat equivalent to just predict whether a bacteria is resistance or not. The resistance mechanisms for these two antibiotics are well studied and have been found to be rather easy to detect, often depending on the presence or absence of certain genes rather than single mutations[46, 47, 48, 49]. Detecting genes can be assumed to be an easier task compared to detecting single mutation within the genome for the neural network.

Moreover, for amoxicillin we found in the literature that the resistance mechanisms seems to be more complex and depending on several genetic factors compared to ampillicin and tetracycline[50, 51, 52, 53, 54]. The DeepMICs performance for amoxicillin is therefore interesting and illustrates the neural networks ability to capture non-linear patterns in our feature space. In figure 5.7 we illustrate how the network detects amoxicillin resistance and we can clearly see that the network is able to group resistant samples together.

Nevertheless, even though the 3-class accuracy is remarkably high for the DeepMIC model, the very major error-rate is over the minimum recommendation for

all antibiotics. An explanation to this might be the bag-of-word approach which neglects the context of k-mers present in the genome and only considers the count. Hence, the model does not account for in which gene a certain k-mer appear.

Additionally, due to the high-dimensional feature space and the many non-linear mappings involved in our model there is no easy way to explain the output of the model. This becomes a drawback for our model since we cannot justify a output, which makes it inadequate for clinical usage. To extend our work and to interpret the DeepMIC model one can investigate the clusters visualized in figure 5.7 and explore how clusters disappears and changes throughout the different parts of the network. Linking the result back to the corresponding DNA-sequences might give an indication of which genes the DeepMIC model base its decision on.

A more direct approach would be to use *shapley sampling* to compute importance for each kmer in the feature space before dimensionality reduction. This approach will however require extensive computational resources in order to before a sufficient number of shapley samplings for each k-mer.

Further, we propose a synthetic minority oversampling approach to generate more training data. However, more studies is needed in order to verify whether this is a good approach to generate synthetic data. A possible limitation with our approach of generating data is that the method is build around generating randomly drawn data-points on the line between two data-points in a neighborhood of each other with the same MIC-value. This will only enhance already existing structures in the annotated dataset and if the neighborhood is defined to be too large, we will generate artificial structures.

Furthermore, we applied semi-supervised learning and implemented a ladder network in order to utilize unlabeled data. This approach was not found to boost the accuracy of the model and increasing the importance of the unlabeled data was found to instead decrease the models accuracy. One reason for this might be the fact that the Salmonella genome has a normal length of over 4 000 000 nucleotides and encodes an enormous amount of phenotypes for each bacteria. Antibiotic resistance is only one of these phenotypes and research have found that antibiotic resistance can be linked to just one or a few genes and sometimes just a single mutation [48]. Despite this, semi-supervised learning is still an interesting approach to investigate, especially in combination with a sequence based model. One could think of a model learning the structure of different genes by using unlabeled data and then learning the resistance mechanisms within these genes using a labeled dataset.

6.2 Limitations with Machine Learning

A limitation with training a machine learning model for prediction of antibiotic resistance is the small datasets in comparison with the large feature space. In this thesis we have 4964 annotated bacteria genomes in contrast to a feature space with 4^{10} dimensions without dimensionality reduction and about 2600 dimensions after PCA based dimensionality reduction. Learning very complex patterns in the genome is therefore very unlikely.

Another limitation with training a machine learning model, within this domain, is the skewness in the dataset. There exists a bias towards sequencing resistant

bacteria since they are more intresting from a clinical perspective which leads to a difficulty for a machine learning model to learn the "standard" genome. This is illustrated by the good performance of our baseline model.

Besides, antibiotic resistance is an always changing landscape. New types of antibiotics results in new types of antibiotic resistances and this dynamics makes building robust prediction models a challenge.

6.3 Future Work

For future work we first propose a sequence based approach using some reccurent neural network structure. This will not cause a loss in information like the bag-of-word approach do and taking the context of k-mers into account might yield better results. This approach is however more data thirsty than a bag-of-word approach and hence larger datasets are needed. Therefore, we still believe that semi-supervised learning is intresting with this domain. One can vaguely think of a model that uses unlabeled data in order to learn genes, since genes are structures that probably clusters well in a feature space, and then utilize labeled data in order to find resistance mechanisms within these genes. In more detail, a *deep belief network* or *reccurent ladder network* might be appropriate for this purpose.

Additionally, our visualization of the data shows that structures occurs within the dataset both after the pre-processing step but also within the latent spaces of the neural network. An emerging field is *topological data analysis* and applying this to the genome of a bacteria in a search for topological structures is an interesting research question.

Further, antibiotic resistance is a always changing landscape. Even if the cost of sequencing DNA would be zero is it possible to easily acquire a diverge set of antibiotic resistant bacteria towards a new type of antibiotics. Thus, in order to build robust machine learning models that can handle the dynamics of evolution *transfer learning* is needed. Transfer learning aims to transfer knowledge from some learning problem into a new but related problem.

Moreover, *generative adverisal models* (GANS) have lately gain success when it comes to generate artificial faces[1] [18, p. 606-609]. Using GANs to generate realistic data is a appealing idea. Even if mutations are assumed to be uniformly distributed over the genome the observed mutations within a population are probably not uniformly distributed since a mutation might alter processes in the bacteria. Advantageous mutation might gain a foothold in the population while disadvantageous ones does not. Hence, the resulting distribution might be complex due to these latent factors. GANs are well suited for modelling complex probability distributions and an intresting research question is whehter it is possible to use them to generate realistic DNA-sequences.

[1] https://thispersondoesnotexist.com/

7
Final Words

From this thesis work we can conclude that machine learning is prominent approach for prediction of antibiotic resistance.

However, several problems are yet to be solve before autonomous prediction models can be implemented in a clinical setting. Therefore, if you want to predict resistance today we recommend a curated database approach since machine learning for antibiotic resistance is not fully mastered and understood today. But if you want to predict antibiotic resistance tomorrow we think that a mixture of machine learning models and curated databases is the way to go. This since they are the opposite of each other, a curated database is a *memory* model where we memorize everything but do not learn from it. In machine learning we instead try to generalize from the data at hand and thus have the possibility to learn the underlying structure of the problem. Combining them would give the predictive power of a machine learning model and the explainability and robustness of a curated database.

Bibliography

[1] WHO, "Ten threats to global health in 2019."

[2] N. Rosenbratt-Farrell, "The Landscape of Antibiotic Resistance," *Environ. Health Perspect.*, vol. 117, no. 6, 2009.

[3] D. Satana, A. Y. Coban, and M. Uzun, "Testing susceptibility of multidrug-resistant Mycobacterium tuberculosis to second-line drugs by use of blood agar," *J. Clin. Microbiol.*, vol. 48, no. 11, pp. 4291–4293, 2010.

[4] E. Zankari, H. Hasman, S. Cosentino, M. Vestergaard, S. Rasmussen, O. Lund, F. M. Aarestrup, and M. V. Larsen, "Identification of acquired antimicrobial resistance genes," *J. Antimicrob. Chemother.*, vol. 67, no. 11, pp. 2640–2644, 2012.

[5] C. M-Bishop, *Pattern Recognition and Machine Learning*.

[6] D. Veltri, U. Kamath, and A. Shehu, "Deep learning improves antimicrobial peptide recognition," *Bioinformatics*, vol. 34, no. 16, pp. 2740–2747, 2018.

[7] G. Arango-Argoty, E. Garner, A. Pruden, L. S. Heath, P. Vikesland, and L. Zhang, "DeepARG: A deep learning approach for predicting antibiotic resistance genes from metagenomic data," *Microbiome*, vol. 6, no. 1, pp. 1–15, 2018.

[8] "https://blogs.plos.org/dnascience/files/2013/07/double-helix.jpg."

[9] National Institute of General Medical Sciences, "National Institute of General Medical Sciences Publicatoins: The New Genetics," no. 10-662, pp. 4–6, 2010.

[10] R. H. Tamarin, *Principles of Genetics*. McGrawHill, The Companies, 7th ed., 2001.

[11] M.Wahde, *Biologically Inspired Optmiziation Methods: An Introduction*. WIT Press, 2008.

[12] S. Behjati and P. S. Tarpey, "What is next generation sequencing?," *Arch. Dis. Child. Educ. Pract. Ed.*, vol. 98, no. 6, pp. 236–238, 2013.

[13] C. Leonard, *Alexander Fleming, 1881-1955*. 1956.

[14] P. M. Hawkey and A. M. Jones, "The changing epidemiology of resistance," *J. Antimicrob. Chemother.*, vol. 64, no. SUPPL.1, 2009.

[15] "Charles C. Brinton's research."

[16] I. Laboratories Inc, "Microbiology guide to interpreting minimum inhibitory concentration," no. Mic, 2017.

[17] C. FDA, USDA, "2010 Narms Executive report," tech. rep., 2010.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. *http://www.deeplearningbook.org*.

[19] G.Cybenko, "Approximation of functions of finite variation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, no. 2, pp. 303–314, 1989.

[20] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," vol. 2, pp. 359–366, 1989.
[21] "compactly supported continuous functions are dense in Lp,"
[22] A. Kathuria, "Intro to optimization in deep learning: Momentum, RMSProp and Adam."
[23] G. E. Hinton, "Optimization: How to make the learning go faster," *Coursera*, vol. 4, pp. 26—-31, 2012.
[24] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015.
[25] R. S. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *http://jmlr.org/papers/v15/srivastava14a.html*, 2014.
[26] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," vol. 48, 2015.
[27] S. Yadav, "Weight Initialization Techniques in Neural Networks."
[28] A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko, "Semi-Supervised Learning with Ladder Networks," 2015.
[29] "By Chervinskii - Own work, CC BY-SA 4.0,"
[30] M. Pezeshki, L. Fan, P. Brakel, A. Courville, and Y. Bengio, "Deconstructing the Ladder Network Architecture," vol. 48, 2015.
[31] J. Brownlee, "A Gentle Introduction to the Bag-of-Words Model."
[32] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.
[33] "Clustering on the output of t-SNE."
[34] A. Ghodsi, "Dimensionality Reduction A Short Tutorial," p. 3, 2006.
[35] P. Method and F. O. R. Approximating, "Power Methods for Approximating Eigenvalues," pp. 586–594.
[36] M. Nguyen, S. Wesley Long, P. F. McDermott, R. J. Olsen, R. Olson, R. L. Stevens, G. H. Tyson, S. Zhao, and J. J. Davis, "Using machine learning to predict antimicrobial MICs and associated genomic features for nontyphoidal Salmonella," *J. Clin. Microbiol.*, vol. 57, no. 2, 2019.
[37] Y. Peng, H. C. Leung, S. M. Yiu, and F. Y. Chin, "IDBA-UD: A de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth," *Bioinformatics*, vol. 28, no. 11, pp. 1420–1428, 2012.
[38] Food and Drug Administration, "Class II Special Controls Guidance Document: Antimicrobial Susceptibility Test (AST) Systems," pp. 1–42, 2009.
[39] M. Kokot, M. Dlugosz, and S. Deorowicz, "KMC 3: counting and manipulating k-mer statistics," *Bioinformatics*, vol. 33, no. 17, pp. 2759–2761, 2017.
[40] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
[41] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: A Python library for model selection and hyperparameter optimization," *Comput. Sci. Discov.*, vol. 8, no. 1, 2015.
[42] T. Klikauer, "Scikit-learn: Machine Learning in Python," *TripleC*, vol. 14, no. 1, pp. 260–264, 2016.

[43] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[44] M. Abadi, A. Agarwal, E. B. Paul Barham, A. D. Zhifeng Chen, Craig Citro, Greg S. Corrado, I. G. Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Y. J. Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, M. S. Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, J. S. Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, P. T. Benoit Steiner, Ilya Sutskever, Kunal Talwar, F. V. Vincent Vanhoucke, Vijay Vasudevan, M. W. Oriol Vinyals, Pete Warden, Martin Wattenberg, Yuan Yu, and X. Zheng., "TensorFlow: Large-scale machine learning on heterogeneous systems," *Methods Enzymol.*, vol. 101, no. C, pp. 582–598, 1983.

[45] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online; accessed <today>].

[46] A. A. S. B S Speer, N B Shoemaker, "Bacterial resistance to tetracycline: mechanisms, transfer, and clinical significance.," *Clin. Microbiol. Rev.*, vol. 5, no. 4, pp. 387–399, 1992.

[47] L. Brin, M. Zarazaga, Y. Sa, F. Ruiz-larrea, and C. Torres, "Beta-Lactamases in Ampicillin-Resistant," *Society*, vol. 46, no. 10, pp. 3156–3163, 2002.

[48] J. M. Munita1 and C. A. Arias, "Mechanisms of Antibiotic Resistance," *Microbiol Spectr.*, vol. 4, no. 2, pp. 1–37, 2016.

[49] R. Cabrera, J. Ruiz, F. Marco, I. Oliveira, M. Arroyo, A. Aladueña, M. A. Usera, M. T. Jiménez De Anta, J. Gascón, and J. Vila, "Mechanism of resistance to several antimicrobial agents in Salmonella clinical isolates causing traveler's diarrhea," *Antimicrob. Agents Chemother.*, vol. 48, no. 10, pp. 3934–3939, 2004.

[50] J. Wang, H. Guo, C. Cao, W. Zhao, L. Y. Kwok, H. Zhang, and W. Zhang, "Characterization of the adaptive amoxicillin resistance of Lactobacillus casei Zhang by proteomic analysis," *Front. Microbiol.*, vol. 9, no. FEB, pp. 1–9, 2018.

[51] R. A. Giannella, S. A. Broitman, and N. Zamcheck, "Salmonella enteritis," *Am. J. Dig. Dis.*, vol. 16, no. 11, pp. 1000–1006, 2005.

[52] C. Llanes, V. Kirchgesner, and P. Plesiat, "Propagation of TEM- and PSE-type β-lactamases among amoxicillin- resistant Salmonella spp. isolated in France," *Antimicrob. Agents Chemother.*, vol. 43, no. 10, pp. 2430–2436, 1999.

[53] L. M. Langata, J. M. Maingi, H. A. Musonye, J. Kiiru, and A. K. Nyamache, "Antimicrobial resistance genes in Salmonella and Escherichia coli isolates from chicken droppings in Nairobi, Kenya," *BMC Res. Notes*, vol. 12, no. 1, pp. 1–6, 2019.

[54] A. Ortega, J. Oteo, M. Aranzamendi-Zaldumbide, R. M. Bartolomé, G. Bou, E. Cercenado, M. C. Conejo, J. J. González-López, M. Marín, L. Martínez-Martínez, M. Merino, F. Navarro, A. Oliver, Á. Pascual, A. Rivera, J. Rodríguez-Baño, I. Weber, B. Aracil, and J. Campos, "Spanish Multicenter Study of the Epidemiology and Mechanisms of Amoxicillin-Clavulanate Resistance in Escherichia coli," *Antimicrob. Agents Chemother.*, vol. 56, no. 7, pp. 3576–3581, 2012.

A

Appendix 1

A.1 t-SNE plots for each antibiotic

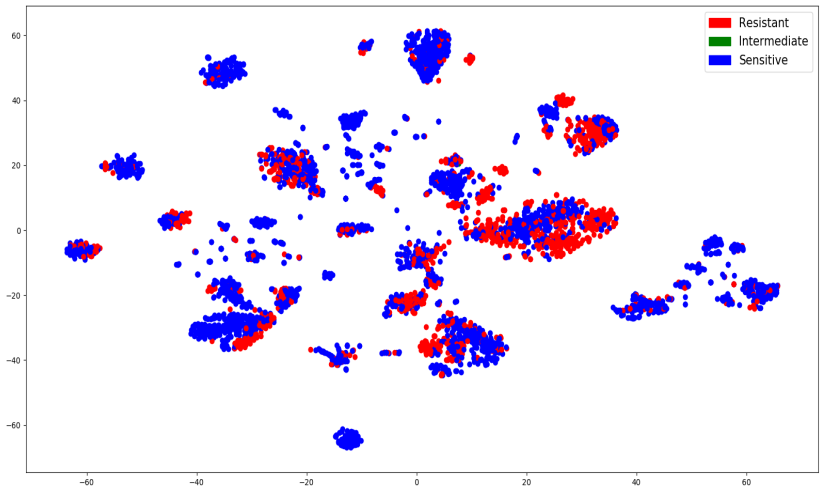


Figure A.1: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect to ampillicin resistance

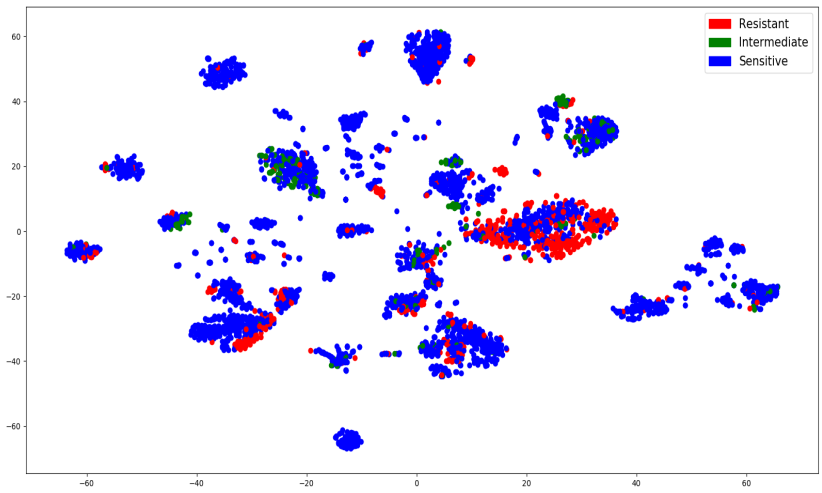


Figure A.2: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect to amoxicillin resistance

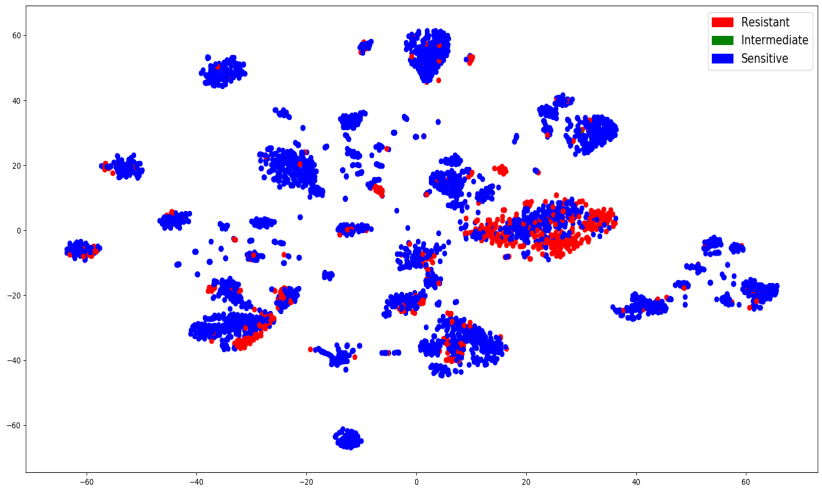


Figure A.3: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect to ceftriaxone resistance

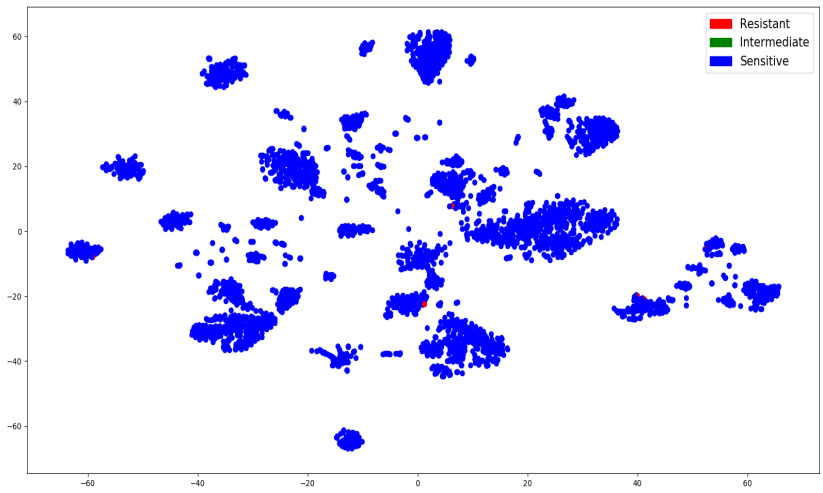


Figure A.4: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect to azithromycin resistance

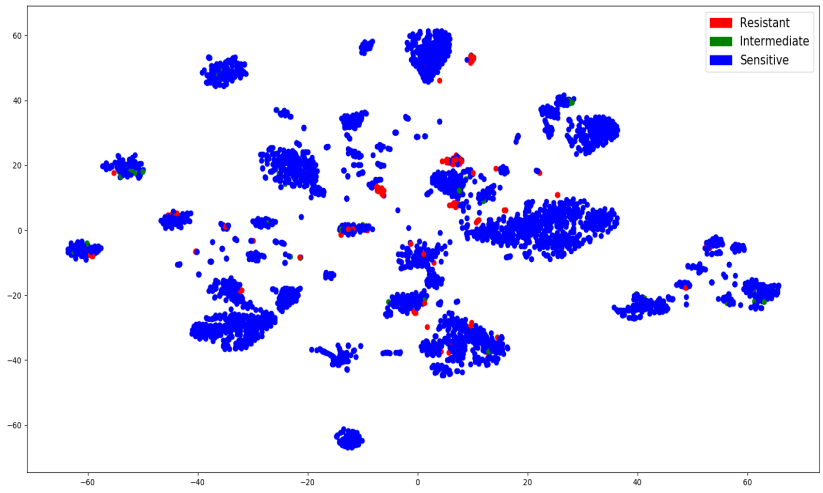


Figure A.5: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect to chlotamphenicol resistance

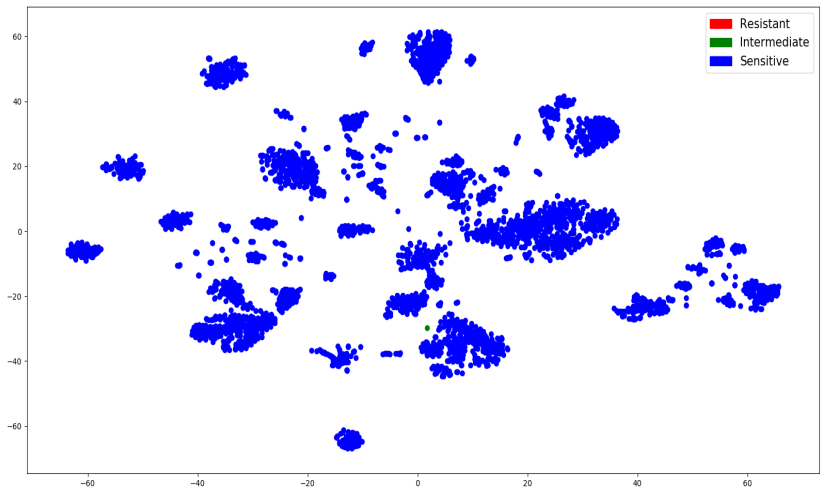


Figure A.6: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect ciprofloaxin resistance

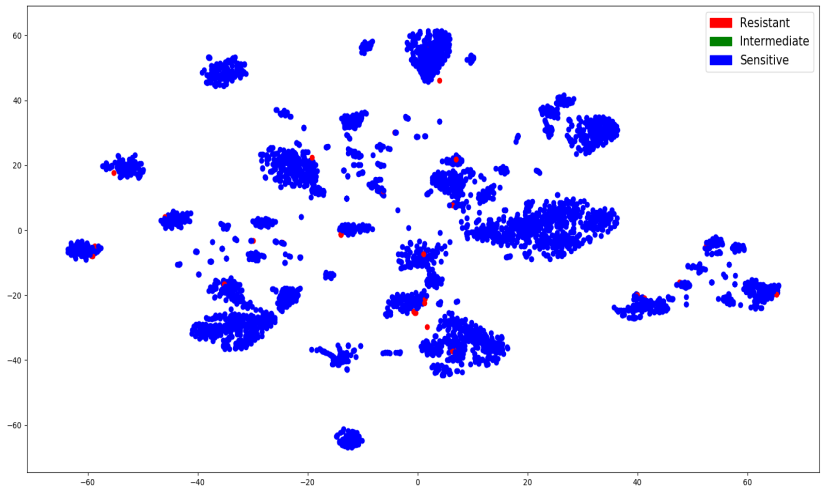


Figure A.7: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect trimethoptim resistance

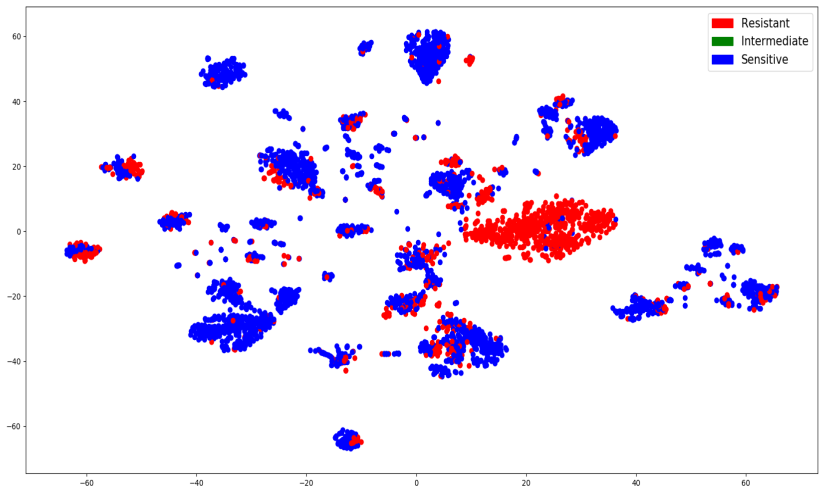


Figure A.8: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect sulfidoxazole resistance

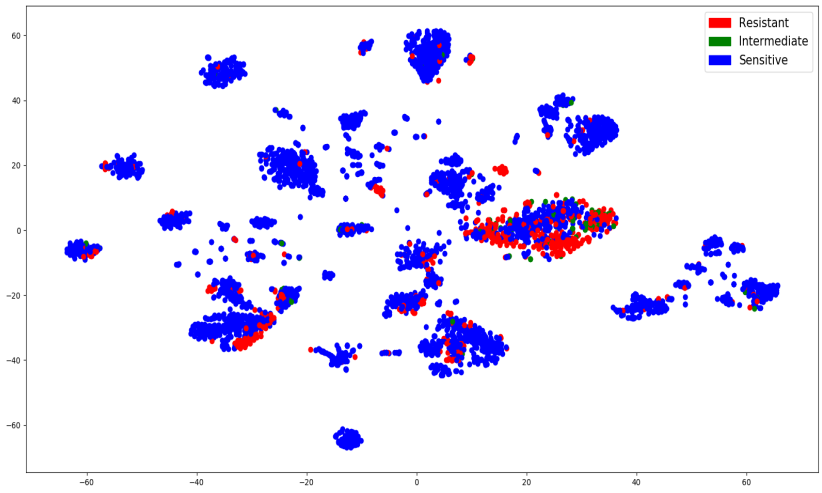


Figure A.9: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect cefoxitin resistance

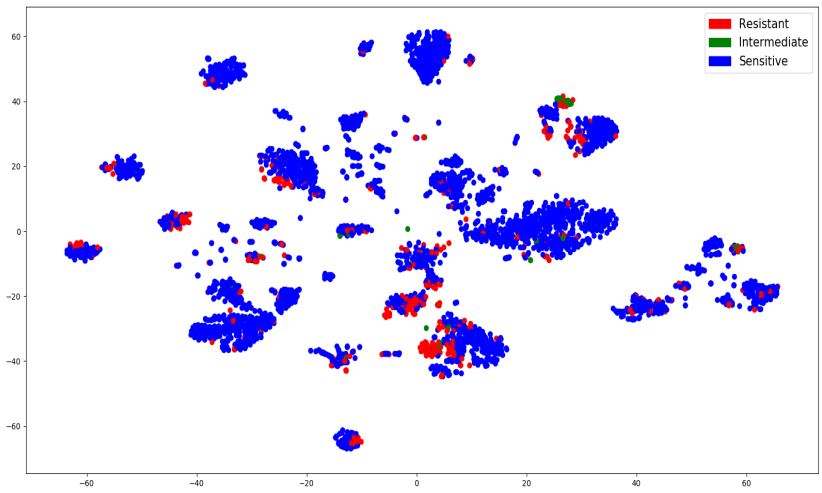


Figure A.10: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect gentamicin resistance

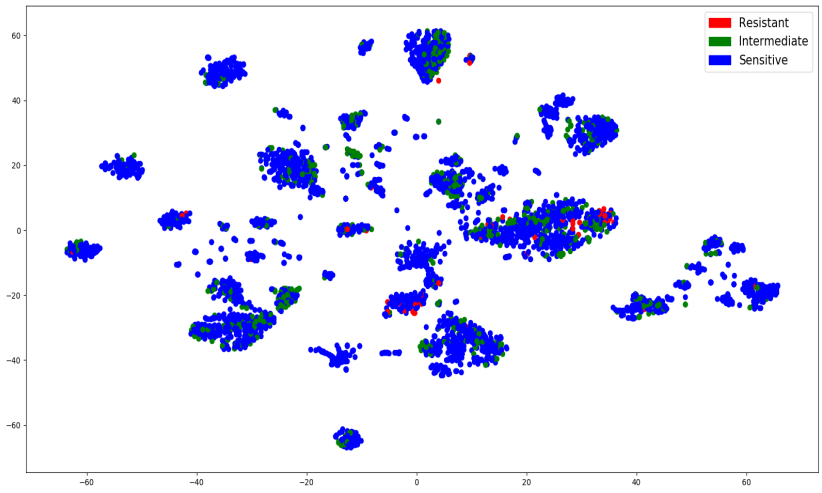


Figure A.11: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect kanamycin resistance

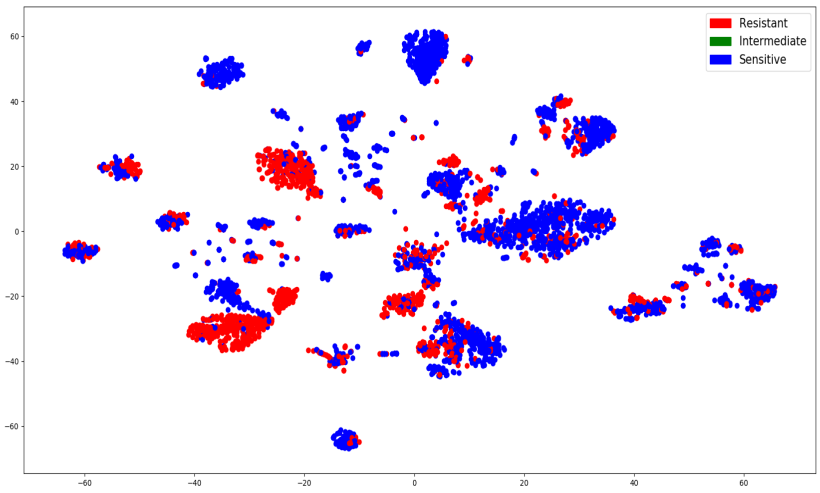


Figure A.12: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect streptomycin resistance

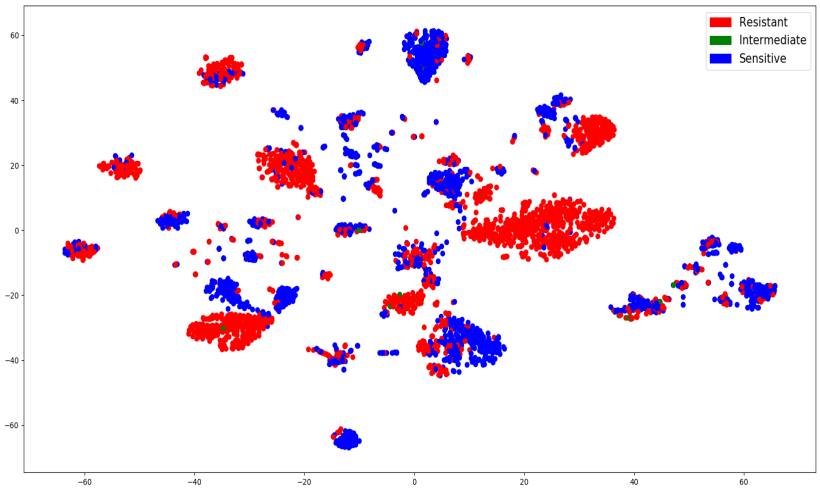


Figure A.13: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect tetracycline resistance

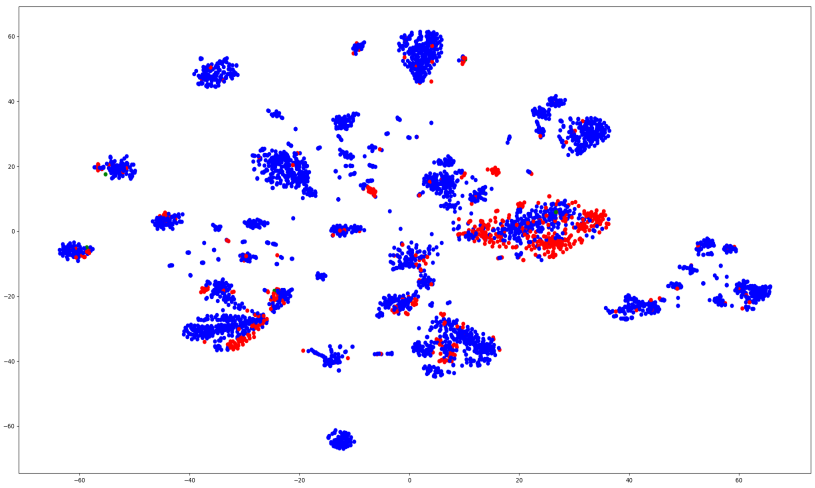


Figure A.14: Annotated data visualized using the manifold learning-algorithm t-SNE and each datapoint colored with respect ceftiofur resistance

A.2 3-class Accuracy and ±1 2-fold Dilution Accuracy

	No Sampling model		Bootstrap model		SMOTE model	
Antibiotics	Mean Accuracy	95% Confidence	Mean Accuracy	95% Confidence	Mean Accuracy	95% Confidence
AMP	0.95	[0.94-0.96]	0.95	[0.94-0.96]	0.94	[0.93-0.95]
AUG	0.94	[0.94-0.96]	0.94	[0.93, 0.95]	0.94	[0.93, 0.95]
AXO	0.98	[0.98-0.98]	0.98	[0.98, 0.98]	0.98	[0.98, 0.98]
AZI	1.00	[0.99-1.00]	1.00	[0.99-1.00]	1	[1.00-1.00]
CHL	0.97	[0.97-0.97]	0.97	[0.97-0.97]	0.97	[0.97-0.97]
CIP	1.00	[1.00-1.00]	1.00	[1.00-1.00]	1	[1.00-1.00]
COT	0.99	[0.99-0.99]	0.99	[0.99-0.99]	0.99	[0.99-0.99]
FIS	0.96	[0.95-0.97]	0.97	[0.96-0.98]	0.96	[0.96-0.96]
FOX	0.96	[0.96-0.96]	0.96	[0.96-0.96]	0.96	[0.96-0.96]
GEN	0.96	[0.96-0.96]	0.95	[0.95-0.95]	0.96	[0.96-0.96]
KAN	0.94	[0.93-0.95]	0.93	[0.92-0.94]	0.94	[0.92-0.96]
NAL	0.99	[0.99-0.99]	0.99	[0.99-0.99]	0.99	[0.99-0.99]
STR	0.92	[0.91-0.93]	0.92	[0.91-0.93]	0.92	[0.90-0.94]
TET	0.95	[0.95-0.95]	0.95	[0.95-0.95]	0.95	[0.95-0.95]
TIO	0.98	[0.98-0.98]	0.98	[0.98-0.98]	0.98	[0.98-0.98]

Table A.1: 3-class accuracy for the oversampling methods and the network trained without oversampling

	Baseline		XGBoost		DeepMIC	
Antibiotics	Mean Accuracy	95% CI	Mean Accuracy	95% CI	Mean Accuracy	95% CI
AMP	0.69	[0.69-0.69]	0,92	[0.90-0.93]	0.94	[0.93-0.95]
AUG	0.69	[0.69-0.69]	0,93	[0.93-0.94]	0.93	[0.93-0.93]
AXO	0.85	[0.85-0.85]	0,95	[0.95-0.96]	0.97	[0.97-0.97]
AZI	0.99	[0.99-0.99]	0,97	[0.96-0.98]	0.99	[0.98-1]
CHL	0.96	[0.96-0.96]	0,99	[0.98-0.99]	0.99	[0.99-0.99]
CIP	0.82	[0.82-0.82]	0,97	[0.97-0.98]	0.92	[0.91-0.93]
COT	0.98	[0.98-0.98]	0,98	[0.97-0.98]	0.98	[0.98-0.98]
FIS	0.33	[0.33-0.33]	0,95	[0.95-0.96]	0.92	[0.91-0.93]
FOX	0.82	[0.82-0.82]	0,96	[0.96-0.97]	0.96	[0.95-0.97]
GEN	0.86	[0.86-0.86]	0,91	[0.90-0.92]	0.95	[0.95-0.95]
KAN	0.90	[0.90-0.90]	0,98	[0.97-1.00]	0.94	[0.92-0.97]
NAL	0.99	[0.99-0.99]	0,96	[0.95-0.97]	0.99	[0.99-0.99]
STR	0.63	[0.63-0.63]	0,93	[0.92-0.94]	0.91	[0.89-0.91]
TET	0.55	[0.55-0.55]	0,90	[0.90-0.91]	0.96	[0.96-0.96]
TIO	0.85	[0.85-0.85]	0,99	[0.99-0.99]	0.98	[0.98-0.98]

Table A.2: Comparison between DeepMIC, the baseline and the XGBoost model w.r.t ±1 2-fold dilution accuracy

B

Appendix 2

B.1 Backpropagation on Matrix-form

In a gradient descent algorithm we iteratively update our solution with respect to the first order derivative. For a feedforward neural network this becomes [18, pp.200-209]

$$\boldsymbol{W}_{l-1,l}^{t+1} = \boldsymbol{W}_{l-1,l}^{t} - \boldsymbol{\eta}_{l-1,l}^{t} \odot \frac{\partial L}{\partial \boldsymbol{W}_{l-1,l}^{t}}, \tag{B.1}$$

$$\boldsymbol{\theta}_{l}^{t+1} = \boldsymbol{\theta}_{l}^{t} - \boldsymbol{\eta}_{l}^{t} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_{l}^{t}}, \tag{B.2}$$

where, $\boldsymbol{\eta}$ denotes the learning rate vector, or updating factors.

Given a loss function L, the derivatives of L w.r.t. the parameters of the feedforward neural network can easily be obtained by exploiting the chain-rule. For weights $\boldsymbol{W}_{l-1,l}$ between arbitrary layers in the feedforward neural network we have

$$\frac{\partial L}{\partial \boldsymbol{W}_{l-1,l}} = \frac{\partial L}{\partial f_{\boldsymbol{w},\boldsymbol{\theta}}} \odot \frac{f_{\boldsymbol{w},\boldsymbol{\theta}}}{\partial \boldsymbol{W}_{l-1,l}}. \tag{B.3}$$

We compute the last derivative

$$\frac{\partial f_{\boldsymbol{w},\boldsymbol{\theta}}}{\partial \boldsymbol{W}_{l-1,l}} = \frac{\partial \sigma(\boldsymbol{z}_L)}{\partial \boldsymbol{W}_{l-1,l}} = \sigma^{'}(\boldsymbol{z}_L) \odot \frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{W}_{l-1,l}} = \sigma^{'}(\boldsymbol{z}_L) \odot \frac{\partial (\boldsymbol{W}_{L-1,L}\boldsymbol{h}_{L-1} + \boldsymbol{\theta}_L)}{\partial \boldsymbol{W}_{l-1,l}}. \tag{B.4}$$

If $L = l$ we get

$$\frac{\partial (\boldsymbol{W}_{L-1,L}\boldsymbol{h}_{L-1} + \boldsymbol{\theta}_L)}{\partial \boldsymbol{W}_{l-1,l}} = \boldsymbol{h}_{L-1} \tag{B.5}$$

otherwise,

$$\frac{\partial (\boldsymbol{W}_{L-1,L}\boldsymbol{h}_{L-1} + \boldsymbol{\theta}_L)}{\partial \boldsymbol{W}_{l-1,l}} = \boldsymbol{W}_{L-1,L}^{T} \odot \frac{\partial \sigma(\boldsymbol{z}_{L-1})}{\partial \boldsymbol{W}_{l-1,l}} = \sigma^{'}(\boldsymbol{z}_{L-1}) \odot \frac{\partial \boldsymbol{z}_{L-1}}{\partial \boldsymbol{W}_{l-1,l}} \tag{B.6}$$

This holds for arbitrary q, l satisfying $l < q$

$$\frac{\partial (\boldsymbol{W}_{q-1,q}\boldsymbol{h}_{q-1} + \boldsymbol{\theta}_q)}{\partial \boldsymbol{W}_{l-1,l}} = \boldsymbol{W}_{q-1,q}^{T} \odot \frac{\partial \sigma(\boldsymbol{z}_{q-1})}{\partial \boldsymbol{W}_{l-1,l}} = \boldsymbol{W}_{q-1,q}^{T} \sigma^{'}(\boldsymbol{z}_{q-1}) \odot \frac{\partial \boldsymbol{z}_{q-1}}{\partial \boldsymbol{W}_{l-1,l}} \tag{B.7}$$

and for $l = q$ we have

$$\frac{\partial (\boldsymbol{W}_{l-1,q}\boldsymbol{h}_{l-1} + \boldsymbol{\theta}_l)}{\partial \boldsymbol{W}_{l-1,l}} = \boldsymbol{h}_{l-1}^{T} \tag{B.8}$$

In the case of $l > q$ we notice that $\boldsymbol{h}_q$ is not depending on $\boldsymbol{W}_{l-1,l}$ and hence

$$\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}_{l-1,l}} = \mathbf{0} \tag{B.9}$$

For a more compact notion we set

$$\boldsymbol{\delta}_L = \frac{\partial L}{\partial f_{\boldsymbol{w},\boldsymbol{\theta}}} \odot \sigma^{'}(\boldsymbol{z}_L) \tag{B.10}$$

$$\boldsymbol{\delta}_l = \boldsymbol{W}_{l,l+1}^T \boldsymbol{\delta}_{l+1} \odot \sigma^{'}(\boldsymbol{z}_l), \quad l < L \tag{B.11}$$

The updating rule for the weights then becomes

$$\boldsymbol{W}_{l-1,l}^{t+1} = \boldsymbol{W}_{l-1,l}^{t} - \boldsymbol{\eta}_{l-1,l}^{t} \odot \boldsymbol{\delta}_l^t (\boldsymbol{h}_{l-1}^t)^T \tag{B.12}$$

Deriving the updating for the biases $\boldsymbol{\theta}_l$ follows by the same argument as for the weights

$$\boldsymbol{\theta}_l^{t+1} = \boldsymbol{\theta}_l^t - \boldsymbol{\eta}_l^t \odot \boldsymbol{\delta}_l^t \tag{B.13}$$

C

Appendix 3

C.1 Genes linked to Antibiotic Resistance in the dataset

Resistance Genes for Ampillicin

1. CARB-1
2. CARB-
3. CMY-130
4. CMY-132
5. CMY-15
6. CMY-16,
7. CMY-2
8. CMY-22
9. CMY-2b
10. CMY-33
11. CMY-94
12. HER-3
13. TEM-1
14. TEM-104
15. TEM-116
16. TEM-117
17. TEM-163
18. TEM-183
19. TEM-192
20. TEM-206
21. TEM-209
22. TEM-214
23. TEM-217
24. TEM-234
25. TEM-30,

Resistance Genes for Streptomycin

1. ant(3")-Ia
2. aph(6)-Ic
3. aph(6)-Id

Resistance Genes for Tetracycline

1. tet(A)
2. tet(B)
3. tet(C)
4. tet(G)
5. tet(M)