Deep learning for prediction of antibiotic resistance genes

Exploration and evaluation of transformer models for metagenomic data

Master's thesis in Engineering Mathematics and Computational Science

ERIKA SALOMONSSON

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Deep learning for prediction of antibiotic resistance genes

Exploration and evaluation of transformer models for metagenomic data

ERIKA SALOMONSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2021

Deep learning for prediction of antibiotic resistance genes
Exploration and evaluation of transformer models for metagenomic data
ERIKA SALOMONSSON

Supervisor: Juan Salvador Inda Díaz, Department of Mathematical Sciences
Supervisor: Erik Kristiansson, Department of Mathematical Sciences
Examiner: Erik Kristiansson, Department of Mathematical Sciences

Master's thesis 2021
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Deep learning for prediction of antibiotic resistance genes
Exploration and evaluation of transformer models for metagenomic data
ERIKA SALOMONSSON
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Antibiotic resistance is a serious public health challenge since it reduces the ability to prevent and treat bacterial infections with antibiotics. Bacteria that are resistant to antibiotics contain resistance genes that are shared between cells. This flow of resistance genes is one of the main reasons behind the rapid global increase of antibiotic resistant bacteria. It is essential to gather information about the already existing resistance genes to be able to counter the flow and to understand what resistance genes might become present in future clinical settings.

The aim of this master's thesis is to investigate if the transformer, which is a relatively new deep learning model, can predict genes that are resistant to the antibiotic class aminoglycosides and also to see if the transformer can distinguish between five different resistance gene classes. An advantage with transformers is that they rely on attention mechanisms that can detect global and complex dependencies in DNA structures which help characterize resistance genes. In order to reach the aim of this project, the architecture and parameters in the transformer model are explored and evaluated to find the model yielding the best performance. The optimal model is then used to make predictions on a real dataset.

We obtained a transformer model that could predict resistance genes with a sensitivity of 0.989 and a specificity of 0.999. Using the same model, around 0.237 % of the real data were predicted as resistant. When the model tried to distinguish between resistance gene classes the sensitivity varied for the classes, where the lowest sensitivity was 0.263 and the highest sensitivity was 0.823. For all classes the specificity was higher than 0.970. A conclusion is that the performance of the transformer model to a great extent depends on the appearance of the input data. The bigger and more diverse dataset, the more dependencies in the DNA structure can be captured implying better performance. With proper datasets the transformer model can make classifications with very good performance.

Keywords: antibiotic resistance, resistance genes, deep learning, transformer model, predictions, sensitivity, specificity.

Acknowledgements

First and foremost, I would like to thank my supervisor Juan Salvador Inda Díaz and my supervisor and examiner Erik Kristiansson for all their engagement and guidance from the start to the end of this master's thesis project. I am thankful for every idea, all feedback and the interesting discussions. Thank you for answering all my questions and for always being just an email away. Without you this work would not have been possible.

Also, many thanks to my family and friends for always being there, supporting me and cheering me up no matter what. You truly mean a lot. To all my Chalmers friends, thanks for all the fun times and for making my five years at Chalmers so great and special. It would not have been the same without you.

Erika Salomonsson, Gothenburg, June 2021

Contents

List of figures

List of tables

1
Introduction

This first chapter begins with a background description that explains the role of antibiotics and antibiotic resistance, and answers the question of why this thesis is carried out. Thereafter, the aim is stated which specifies the intended outcome and lastly an outline of the different sections in this thesis is presented.

1.1 Background

Antibiotics are widely used to prevent and treat many types of bacterial infections. If bacteria become *resistant* to the antibiotics that are supposed to kill them, the antibiotics can no longer be used as a treatment for the infection. Antibiotic resistance decreases the effectiveness of antibiotics and is currently considered as one of the most serious public health challenges [2]. Bacteria that are resistant to antibiotics contain *resistance genes* which are shared between cells. Genes that are resistant to a certain antibiotic can often be divided into several subclasses based on similarities in the gene structure [3]. Apart from resistance genes it also exists *susceptible genes*. Susceptible genes are similar to resistance genes but do not provide antibiotic resistance [4]. The sharing of genes causes a flow of resistance genes from the environment, where most of the clinically essential resistance genes probably originate, via human and animal *commensals* and finally into *pathogens*. Usually, a commensal bacteria does not cause a disease while a pathogenic bacteria does [5]. Due to the flow of resistance genes there is an increase in antibiotic resistance worldwide. It is important to identify the existing resistance genes to be able to counter the flow and to gather information about the resistance genes that eventually might be present in the clinical settings.

Previously, a computational method based on hidden Markov models (HMMs) called *fARGene* has been developed to characterize new resistance genes. The method can identify many new resistance genes with high accuracy even if the genes differ significantly from the already known resistance genes. One limitation of fARGene is that it does not detect all existing dependencies in the genes which implies that some resistance genes remain unidentified [6].

In 2017 a new natural language tool called *the transformer* was introduced. The transformer is a network architecture that relies completely on an *attention mechanism* in order to detect global dependencies between input and output data [1]. Therefore, it is of interest to investigate if a method based on transformers can

identify and use complex dependencies in genes to characterize the resistance genes fARGene is unable to.

1.2 Aim

The aim of this master's thesis is to classify *metagenomic reads*, i.e. random parts of DNA, and to detect new forms of resistance genes by developing and implementing an AI-method based on transformers. More specific, the aim can be divided into three parts:

i. Explore and evaluate the transformer as a model for predicting resistance genes.

ii. Investigate if the transformer model can distinguish between resistance and susceptible genes, but also detect new forms of resistance genes. Preferably, the model should find all resistance genes without classifying any susceptible genes as new forms of resistance genes.

iii. Investigate if the transformer model can distinguish between different resistance gene classes.

1.3 Outline

The outline of the thesis is as follows. Previously in this first introduction section a background of antibiotic resistance, an earlier work and a motivation for why this thesis is of value are presented. To understand the intended outcome of this thesis the aims are also part of the introduction. Section 2 consists of all theory needed. More precisely, essential biological facts are described, an introduction and a comparison between natural language processing tools are available and lastly the transformer model including its architecture and parameters is explained. In Section 3 the methods are stated, which describe the data used in the thesis and also how the transformers are being trained. How the architecture and parameters are explored and evaluated in order to yield the best results based on some performance metrics are also part of the section. Thereafter, the results are presented in Section 4. The results show how the model performances differ depending on the parameter values and input data, as well as the performances of the final models together with predictions of a real metagenomic dataset. Lastly, Section 5 consists of an analysis and a discussion of the data, the transformer model, the obtained results and also some possible future work.

2
Theory

Theory that is essential for this thesis includes some basics about DNA and how it can be analyzed. Moreover, it is of high importance to have sufficient knowledge about the transformer architecture and how the transformer works before it can be implemented, explored and evaluated. Therefore, descriptions of these concepts are provided here.

2.1 Biology

First of all, important biological facts are stated and explained. We begin with a description of DNA followed by a presentation of sequencing platforms used to determine and analyze DNA. The biology section ends with a technique that facilitates DNA analyses.

2.1.1 Basics of DNA

All organisms are unique and it is due to biological instructions that enable an organism to develop, survive and reproduce [7]. The biological instructions are encoded in molecules called *deoxyribonucleic acid* which is more known as DNA. *Nucleotides* are the blocks building up the DNA molecules and each nucleotide consists of a *phosphate group*, a *sugar group* and a *nitrogen base*. There are four nitrogen bases, namely *adenine (A)*, *thymine (T)*, *guanine (G)* and *cytosine (C)*. DNA has a twisted structure like a double helix and it is formed by paring the nucleotides. A pair of nucleotides is called a *base pair* and does always consist of either an A and a T or a C and a G. The biological instructions are decided by the order of the nucleotides in the DNA.

The set of all DNA in an organism is called a *genome* and the study of the genome is called *genomics* [8]. *Metagenomics* is the study of a mix of genomes present in the environment. In the human genome, approximately one percent of the DNA contains biological instructions needed to create *proteins* which are the molecules that actually carry out most of the work [7]. The parts of the DNA that contain these instructions are called *genes* [8].

2.1.2 DNA sequencing

Genetic changes and mutations in DNA can imply certain diseases [8]. Determining and analyzing the order of nucleotides are therefore important, for instance in order to understand what causes a disease and to detect novel genes [9]. The first DNA sequencing method was known as *Sanger sequencing* and was introduced in 1975. Since then there has been a need for faster and more cost-efficient methods which has implied a quick development and resulted in *next generation sequencing*, or *NGS* in short. NGS platforms can sequence millions of parts from DNA in parallel generating high-throughput data which we refer to as *sequencing reads*. Nowadays, several NGS platforms exist and one of them is the *Illumina platform*. The Illumina platform has higher throughput than all other platforms and also the lowest cost per nucleotide. Another advantage with Illumina sequencing is that it can be performed *paired-end*, meaning that parts of DNA are sequenced from both ends. Reads sequenced using Illumina are at most 300 base pairs long [10].

2.1.3 *k*-mers

When analyzing biological data, reads are often divided into *k-mers* which are smaller parts of length k. A bigger k provides more contextual information but is at the same time often more computationally heavy. The k-mers can either be overlapping or non-overlapping. In both cases the number of different k-mers is 4^k where number four refers to the number of unique nucleotides (A, C, G and T). Figure 2.1 shows how non-overlapping 3-mers are produced from a read.

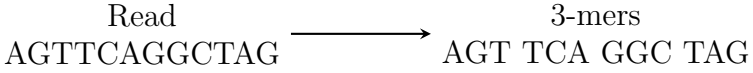

Figure 2.1: An illustration of how a read is divided into non-overlapping 3-mers.

2.2 Natural language processing tools

Natural language processing (NLP) originates from several scientific fields such as computer sciences, linguistics, mathematics and artificial intelligence [11]. NLP focuses on the combination between computers and natural language text and speech. Especially, the aim of NLP is to develop and use computer tools and techniques that can understand natural language and perform tasks such as natural language text processing and summarizing, speech recognition and user interfaces.

There are several natural language processing tools. In the 1980's tools based on statistical approaches, for example HMMs, became popular [12]. HMM methods learn from observed data and produce outputs based on probabilities. In the case when HMM methods are used to predict sentences, each word in the sentence has a probability and it is assumed that each word only depends on the words before it. One problem with this approach is that the computational time increases massively as the sentences become longer.

A natural language processing tool that is widely used today is recurrent neural networks (RNNs) [13]. RNNs consist of hidden states h_t at time step t which are updated depending on the hidden states h_{t-1} at time step $t-1$ together with the input data x_t at time t. Since the data is processed in sequential order parallelization is not possible which can lead to high computational times. RNNs also suffer from the vanishing gradient problem which makes it difficult to learn and tune parameters in early network layers. Therefore, it can be hard to find dependencies between parts of the input data that are far away from each other in the sequence.

The previous problem with the lack of parallelization and ability to find all dependencies between input and output data no matter the distance, gave in 2017 rise to the new deep learning architecture called the transformer [1]. Some differences between RNNs and the transformer are that the transformer relies completely on an attention mechanism to detect global and complex dependencies, and does not make use of recurrence at all. Also, computations in the transformer can be done in parallel which decreases the computational times. How the attention mechanism and the transformer in general work are described in the following section.

2.3 Transformers

Now, when we have got a motivation for why the transformer model was developed, we will describe the transformer architecture and how the model works.

2.3.1 Overall transformer architecture

At the time when the transformer was introduced it was mainly used for translation. However, the model architecture can be adapted to suit a variety of tasks, for instance to make classifications. In order to classify reads as either from susceptible or resistance genes we can use the transformer model shown in Figure 2.2. As input we feed the read we want to classify and as output we get predictions for the susceptible class and the resistance class.

When an input read is fed into the transformer the first step is to divide the read into either overlapping or non-overlapping k-mers. The splitting is sometimes referred to as *tokenization* and a created smaller part is often referred to as a *token* i, where $i = 1, ..., n$ and n is the number of tokens the read is tokenized into. In Figure 2.2 the input read is divided into 1-mers, i.e. tokens of length one. The tokens are then used to build up the *vocabulary* which consists of all unique tokens that will be considered in the transformer. A start and an end token are also part of the vocabulary to represent the start and the end of the reads. Hence, the number of unique tokens in the vocabulary is $4^k + 2$. The possible class labels are also added to a vocabulary. In this case, the number of class labels is two since we have one label for the resistance class and one label for the susceptible class.

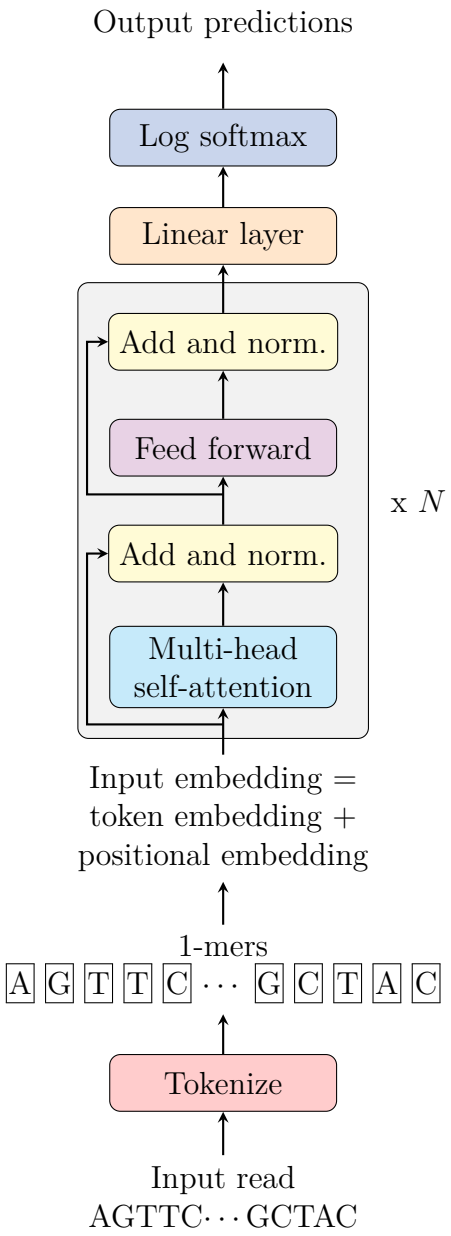

Figure 2.2: An example of an architecture of a transformer model where reads are used as inputs and class predictions are generated as outputs. Here, the input read is divided into 1-mers.

Many machine learning algorithms, including the transformer model, prefer to handle values instead of strings [14]. In the transformer each token in the vocabulary is therefore mapped to a vector of continuous values. These vectors are called *token embeddings* and can be initialized in various ways, for instance randomly. During training the values in the token embeddings are adjusted and tokens that are similar or relevant to each other, according to the transformers, tend to cluster and to be close to each other in the vector space. For simplicity, the token embeddings can be thought of as a lookup table where each token in the input read corresponds to a vector and all tokens that are equal in the input read map to the same vec-

tor. Moreover, we have *positional embeddings* which basically are vectors that keep track of the positions of the tokens in the input read. Positional embeddings can be both learnable and fixed [1]. An *input embedding* x_i for token i is then created by summing the token embedding and the positional embedding. Therefore, all vectors need to be of the same dimension which is referred to as the *embedding size*, d_{model}, and is set by the user. The bigger the embedding size, the more contextual differences can be captured in the data, but if the embedding size is too big there is a risk of overfitting. In such cases, a smaller embedding size is preferable.

When the input embeddings are obtained they are fed into the *encoder* which is the gray block in Figure 2.2. A transformer consists of N identical encoder blocks stacked on each other, where the output from one encoder block is used as input to the consecutive encoder block. The number of encoder blocks is also called *the depth*. A higher number of the depth means a bigger model which should imply better performance. On the other hand, the computational time increases significantly with the depth so a proper trade-off should be made. An encoder contains two main components: *multi-head self-attention* and a *feed forward neural network*. After each of these components there is an *add and normalization layer*. The dimensions of the input and output embeddings of an encoder are the same, and the dimensions are also preserved between all encoder components. In each part of the encoder the embeddings are processed and updated using learnable parameters with the overall aim to obtain information about which tokens in the sequence that are important to each other based on the structure in the input read. The transformer tries to find patterns and dependencies that are typical for each output class. More details about the different parts in the encoder will be provided shortly.

After the embeddings have been processed in the encoders we want to map the embeddings to a log-probability distribution to get predictions of the output classes. Here, the possible outcomes are resistant and susceptible. The outcome predictions are generated using a linear layer which produces log logits to the activation function which usually is the *log-softmax function* defined as

$$\text{LogSoftmax}(y_k) = \log\left(\frac{e^{y_k}}{\sum_{j=1}^{M} e^{y_j}}\right),$$

where y_k is the log logit for class k and M is the number of output classes [15]. The log-softmax function calculates one score for the resistance class and one for the susceptible class and the class with the highest score is the most likely class. These scores could be seen as log-probabilities. If we take the natural exponential function of the scores we get values between zero and one which together sum to one. These natural exponential function values can be interpreted as the class probabilities, i.e. the probabilities for the considered input read to belong to the resistance and susceptible classes.

2.3.2 Multi-head self-attention

Now, when we have a sense of how the transformer works we will dig into the details of the encoder. As previously mentioned, the first component in the encoder is the multi-head self-attention. The multi-head self-attention consists of h *attention heads* that work in parallel [1]. The purpose of an attention head is to weigh all tokens in the sequence against each other and produce updated embeddings with information about the specific token as well as a weighted combination of the other tokens. Tokens that the transformer considers as relevant to the considered token get high weights and thus much focus is placed on these tokens. By using several attention heads different types of relevance and dependencies can be captured since each attention head focuses on a specific part of the embeddings.

Figure 2.3 illustrates the multi-head self-attention thoroughly. The inputs are called queries, keys and values. Since it has been proven preferable to use several smaller attention heads instead of one big, we make h linear projections with different learnable parameters of the queries, keys and values into dimension d. Often, $d = d_{model}/h$ to make the computation of multi-head self-attention almost constant, but other dimensions can also be used [16]. Each token in the input has a query, key and value vector which are equal to the input embedding in the first step. Due to computational reasons, all tokens in the input read are considered at the same time and the queries, keys and values are combined into matrices Q, K and V, respectively [1].

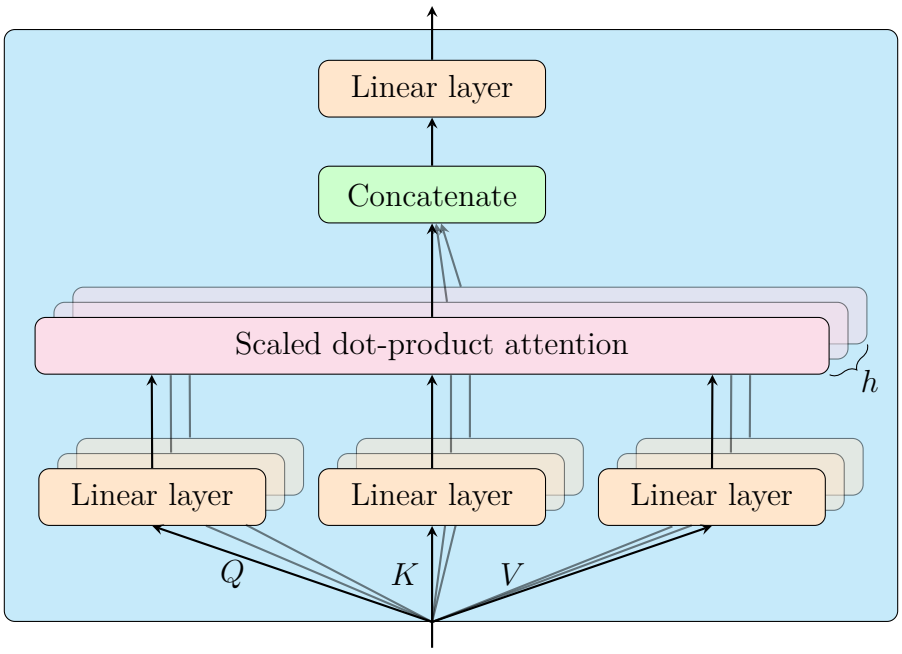

Figure 2.3: Multi-head self-attention uses h parallel attention heads to update the embeddings based on information in the tokens and dependencies between tokens.

Relevance between the tokens in the input read are estimated by calculating *attention weights* between all tokens according to the so called *scaled dot-product attention*. The scaled dot-product attention is performed on the projected versions of the queries, keys and values and is illustrated in detail in Figure 2.4. In mathematical terms the scaled dot-product attention is expressed as

$$\text{Attention}(Q, K, V) = \text{Softmax}\Big(\frac{QK^T}{\sqrt{d}}\Big)V,$$

where the output is a matrix containing updated versions of the embeddings of dimension d. The scaling factor $1/\sqrt{d}$ is included since it is argued that it implies better learning, and the softmax function is used to generate normalized attention weights that are positive and sum to one [17]. By multiplying the attention weights with the value matrix V, the values of the relevant embeddings are preserved since these are multiplied with high weights while the irrelevant embeddings are vanished since these are multiplied with small weights.

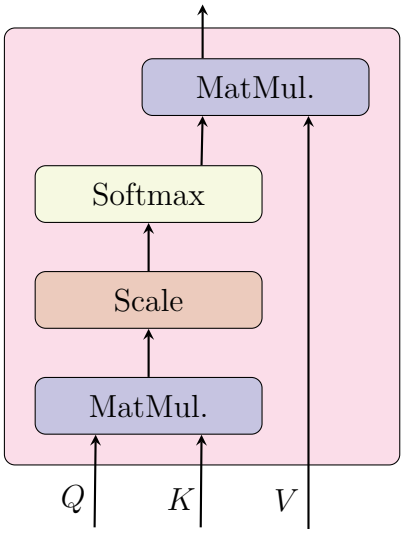

Figure 2.4: The scaled dot-product attention calculates attention weights between all tokens which are used to process and update the embeddings.

After the scaled dot-product attention the outputs from the attention heads are concatenated and linearly projected using learnable parameters. The concatenation and projection are done according to

$$\text{Multi-head}(Q, K, V) = \text{Concatenate}(\text{head}_1, ..., \text{head}_h)W^O.$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

Here, W_i^Q, W_i^K and W_i^V are parameter matrices of dimension d_{model} x d and W^O is a parameter matrix of dimension d_{model} x d_{model}. These parameter matrices are used for the linear projections before and after the scaled dot-product attention [1].

2.3.3 Add and normalization layer

As seen in Figure 2.2, the updated embeddings from the multi-head self attention are fed into an add and normalization layer. Around the multi-head self attention component there is a residual connection. This means that the embeddings before and after the multi-head self-attention are added and then the sum is normalized. The calculations can be summarized as

$$\text{Norm}(\mathbf{x} + \text{multi-head self-attention}(\mathbf{x})).$$

Here, $\mathbf{x} = [x_1, ..., x_n]$ is the sequence of embeddings. One advantage of having the residual connection is that it helps the transformer model remember the positions of the tokens in the read. The residual connection also implies stronger gradients which yields better learning. Normalizing the sum enables larger learning rates and thus faster training. Moreover, the normalization centers the embeddings around the origins which might be helpful when performing self-attention [18].

The last part in the encoder is another add and normalization layer which has the same purpose as the first one, with the only difference that the embeddings now come from the previous add and normalization layer and the feed forward layer.

2.3.4 Feed forward layer

After the first add and normalization layer a fully-connected feed forward layer follows, which is applied to each embedding and is used to further process the embeddings [1]. The feed forward layer consists of two linear layers with a ReLU layer in between as shown in Figure 2.5.

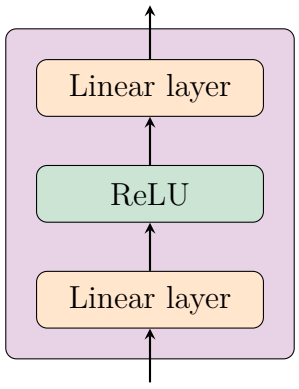

Figure 2.5: The feed forward layer consists of a linear layer, a ReLU layer and another linear layer which is applied to each embedding separately and identically [1].

In mathematical terms, the feed forward layer performs the computations

$$\text{Feed forward}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + b_1)W_2 + b_2.$$

The parameters in the linear layers are the same for all tokens but they vary between encoders.

2.3.5 Possible variations in the transformer model

At this point the transformer model in Figure 2.2 is completely described. As stated earlier, the tokens in this transformer model are 1-mers but they could have been other k-mers as well. It is also possible to extend the transformer model for accepting several different k-mers at the same time and one example of such model is illustrated in Figure 2.6. Here, 1-mers, 2-mers, 3-mers, 4-mers and 5-mers are considered simultaneously.

In the transformer model with multiple k-mers the input read is tokenized for each value of k to create k sequences of tokens of different lengths. Each sequence of tokens is then embedded and processed in a separate stack of N_{k-mers} encoders with their own parameter values. When the sequences of different k-mers have been processed they are concatenated and output predictions for the classes are generated in the same way as in the previous model where only one type of k-mer was used. An advantage of considering several k-mers simultaneously is that more information and different types of relevance in the input read can be captured which should improve the overall performance.

So far, we have assumed that an input read either belongs to a susceptible or a resistance class, but of course it is possible to add and make predictions for more classes. This is achieved by extending the vocabulary with more labels.

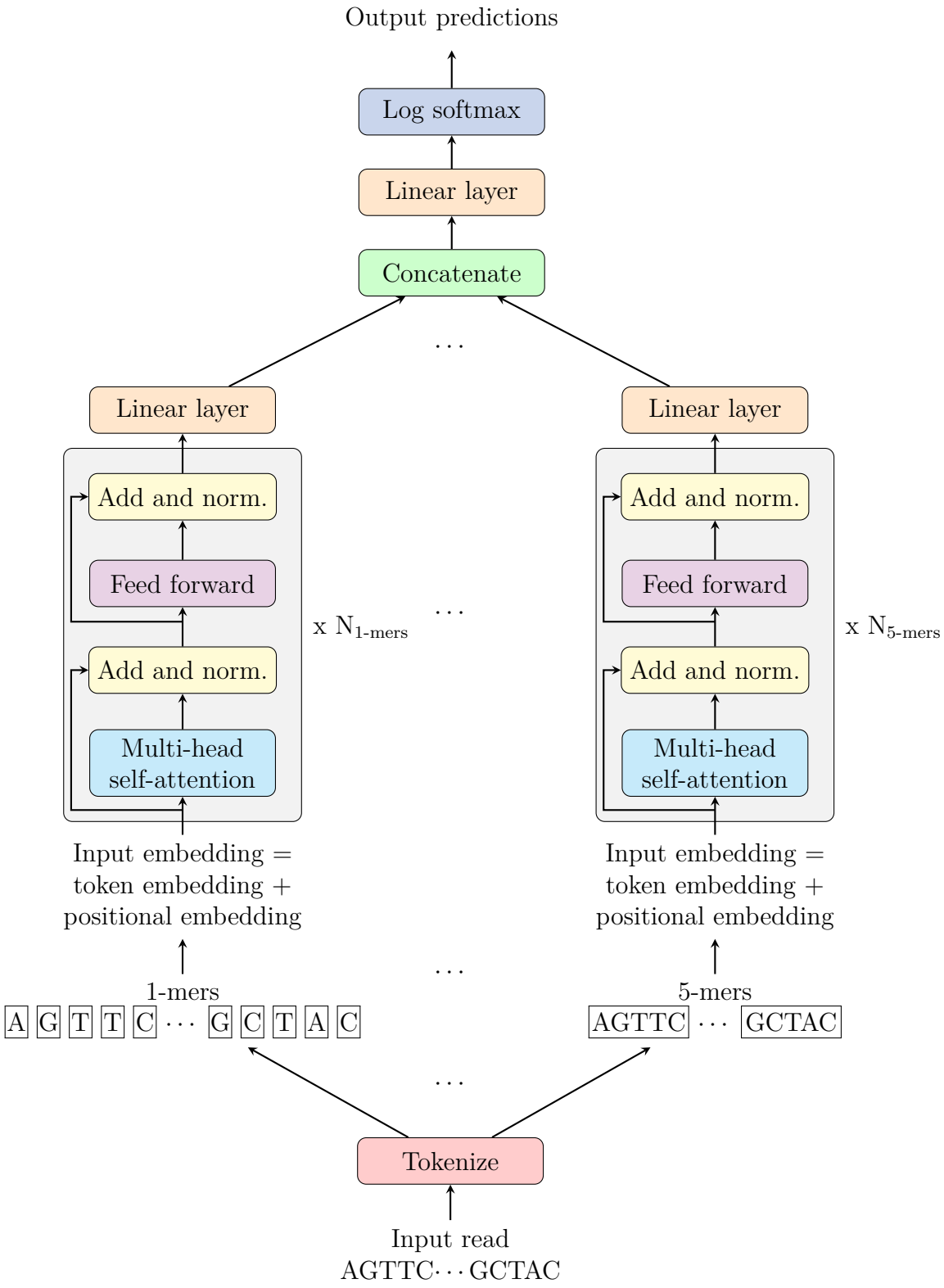

Figure 2.6: An examples of an architecture of a multiple k-mer transformer model, where the input read is tokenized into 1-mer, 2-mers, 3-mers, 4-mers and 5-mers. The tokenized sequences are then embedded and processed in encoders separately before they are concatenated to generate output predictions for the classes.

3
Methods

Now, we will explain the work that has been done in order to reach the aims of this master's thesis. The first part of the method was to create datasets with resistant and susceptible reads. Then, a transformer model was implemented, trained and evaluated with different parameter values and k-mers to find the model yielding the best performance. The optimal model was then used to classify reads from a real metagenomic dataset. As a last part, the transformer model was extended to handle more than two classes and it was investigated if the model could be trained and evaluated to distinguish between different resistance gene classes.

All implementations, training and evaluations of transformer models have been performed using the package PyTorch in Python.

3.1 The data

To be able to train and evaluate a transformer model proper datasets with reads and class labels are needed. In this project we decided to create a model for classifying short metagenomic reads as susceptible or resistant to a class of antibiotics named *aminoglycosides*. At our disposal we had a number of DNA datasets containing genes that are susceptible or resistant to aminoglycosides. Table 3.1 presents the number of genes for three datasets that were used in the project. Two of the datasets in the table contain susceptible genes and are called *susceptible 1* and *susceptible 2*, and one of the datasets contains resistance genes and is therefore called *resistance 1*. As seen, in total there are more susceptible than resistance genes.

Table 3.1: Datasets with susceptible or resistance genes for aminoglycosides. Two of these datasets contain susceptible genes and one of the datasets contains resistance genes.

Dataset	Number of genes
Susceptible 1	374
Susceptible 2	59
Resistance 1	85

Genes that are resistant to aminoglycosides can be arranged into subclasses based on their structure [19]. The resistance gene classes that were considered in this project are *aac2p*, *aac3*, *aac6p*, *aph3p* and *aph6*, where aac3 and aac6p can be arranged into

even smaller subclasses. All datasets with resistance genes from different subclasses that were used in this project are presented in Table 3.2 together with the number of genes in each dataset. For simplicity, the name of the datasets are the same as the name of the resistance gene class they consist of. The resistance gene classes aph3p and aph6 are different in their gene structure in comparison to the other gene classes and have previously been observed as more challenging when finding dependencies and making predictions. Due to that, different susceptible genes were considered depending on which resistance genes that were used. More precisely, the dataset named susceptible 2 was used together with the challenging resistance gene classes, and the dataset named susceptible 1 was used with the rest of the resistance genes.

Table 3.2: There is one dataset for each resistance gene class and the number of genes for each class varies.

Resistance gene class	**Number of genes**
aac2p	5
aac3_class1	8
aac3_class2	21
aac6p_class1	16
aac6p_class2	24
aac6p_class3	18
aph3p	46
aph6	10

Sometimes when training and evaluating transformer models we combined a number of datasets from Table 3.1 and Table 3.2. In other scenarios only one dataset with susceptible genes and one dataset with resistance genes were used.

If the same genes are included in both training and evaluation of a transformer model, the performance will probably be a bit optimistic and therefore all DNA datasets were divided into separate train and validation datasets where 70 % of the genes were used for training and 30 % for validation. The package Bio.SeqIO helps splitting the data in Python. Sequencing reads were then simulated from the train and validation files with a program called *Art*. Art can simulate single-end, paired-end/mate-pair reads of the next-generation sequencing platforms Illumina's Solexa, Roche's 454 and Applied Biosystems' SOLiD [20]. Paired-end Illumina read simulation was used in this project to simulate sequencing reads of 100 nucleotides.

Moreover, two additional datasets were available which already contained reads. The first dataset had 300000 susceptible reads which were split into training and validation datasets with the same ratio as the datasets above. The second dataset consisted of one million *real metagenomic reads* where it was unknown which class the reads belonged to. Therefore, it was of interest to make predictions of the reads in this dataset when an optimal transformer model was found.

3.2 Training

All transformer models in this project were trained using a training dataset and evaluated using a validation dataset. We optimized our models by minimizing the negative log-likelihood loss function. The negative log-likelihood is often used to train classification problems [21]. In this project, the input to the loss function is the output from the log-softmax function in the transformer model which contains scores, or log-probabilities, of each class. We also provided weights to the loss function since our training dataset in all cases were imbalanced, meaning that the number or reads for the classes were not the same. The number of resistant reads was always smaller than the number of susceptible reads. To compensate for this, we assigned a bigger weight to reads from the smaller dataset. For example, assume that we had two classes in our training dataset where the first class consisted of 10000 reads and the second class consisted of 20000 reads. The second dataset is twice as big as the first dataset and therefore we assigned a twice as big weight to the first dataset. Moreover, the weight for the first and second class summed to one. When the training dataset consisted of more classes than two, weights were assigned in a similar way.

The loss function was calculated for a batch of reads, where the number of reads in a batch is known as the batch size. The batch size can be anything from one read to the total number of reads in the dataset. If the batch size is the same as the total number of reads the model is guaranteed to converge to a minimum but a lot of memory is required which may cause computational problems. If the batch size on the other hand is one, the training is very fast but it is not sure that the model will reach a minimum. Hence, a proper value of the batch size should be chosen. In this project, we decided to try two different values of the batch size and then choose the batch size implying the best performance.

In general, when we trained our models the training and validation losses decreased for each training step, also known as an epoch, but after a certain number of epochs the validation loss started increasing. To avoid overfitting our models we stopped the training straight before the validation loss started increasing.

3.3 Basic transformer model

Firstly, the transformer model in Figure 3.1 was implemented in Python and we refer to it as *the basic transformer model*. The basic transformer model takes input reads of 100 nucleotides and generates output predictions for the susceptible class S, and for the resistance class R. We assume that each read belongs to the class with the highest output prediction value. The transformer model can be customized in various ways by selecting values of parameters and choosing and combining different k-mers. However, learnable positional embeddings were always used. How different values of parameters and k-mers affected the performance of the transformer were investigated in order to obtain the optimal model for our task.

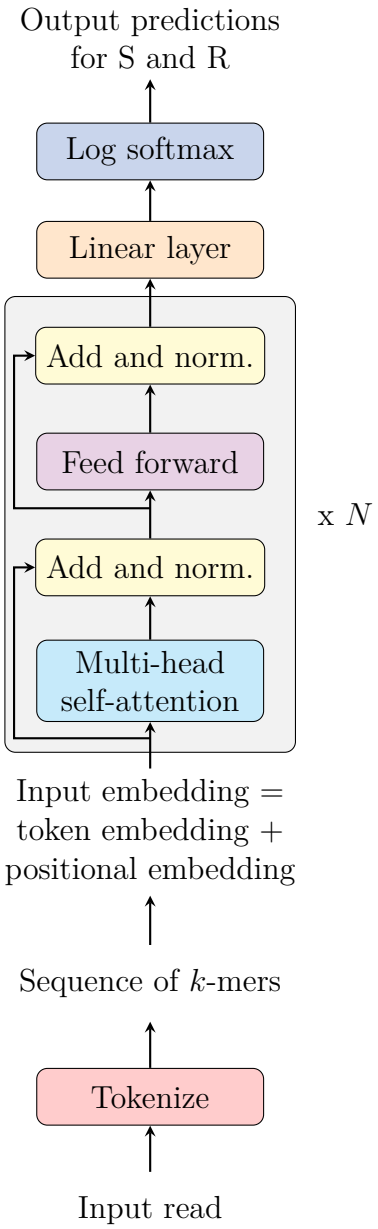

Figure 3.1: The basic transformer model where input reads of length 100 nucleotides are divided into k-mers and class predictions for S and R are generated as outputs.

3.4 Exploration of parameter values

For exploration of parameter values we used the basic transformer model with input reads tokenized as 1-mers. The model was trained and evaluated with reads simulated from the genes in the datasets named susceptible 1 and resistance 1 presented in Table 3.1. We chose these datasets for the simple reason that they were the first datasets available in this project. Table 3.3 shows how many susceptible and resistant reads that were generated for the training and validation datasets.

Table 3.3: The number of susceptible and resistant reads used when training and evaluating the basic transformer model for different parameter values. Here, reads are generated from the datasets susceptible 1 and resistance 1.

	Number of reads	
	S	R
Training	84626	15300
Validation	35750	7150

Parameters that were explored were embedding size, number of heads in the multi-head self-attention, depth and batch size. For each parameter we considered it enough to try two different values to get a sense of how they affect the output from the transformer, and which values to choose in order to achieve the best performance. A comparison between different parameter values was made by running and evaluating the transformer model with a standard setting and then changing one parameter value at a time. More precisely, the values were as in Table 3.4 and the values implying the best performance were used in the model from now on, unless otherwise stated.

Table 3.4: A summary of the values of embedding size, number of heads, depth and batch size in the standard setting. Change 1 to 4 show which additional values were investigated. The values implying the best performance were then chosen.

	Embedding size	**Number of heads**	**Depth**	**Batch size**
Standard	16	2	3	32
Change 1	8			
Change 2		4		
Change 3			6	
Change 4				64

3.5 Exploration of *k*-mers

When the parameter values had been explored and evaluated we got an insight into how well the basic transformer model distinguishes between susceptible and resistant reads, where the resistant reads were from different subclasses. It is also interesting to see how well the basic transformer model performs if the resistant reads are less diverse and more homogeneous in their structure. Therefore, when exploring various k-mers we decided to train and evaluate the basic transformer model on resistant reads simulated from a single resistance gene class. We chose subclass aac6p_class2 since it is the dataset with most genes. Still, we used susceptible reads from the dataset called susceptible 1. The number of reads in the new training and validation datasets are stated in Table 3.5. We compared the performance of the transformer models when non-overlapping 1-mers, 2-mers, 3-mers, 4-mers, 5-mers and 6-mers were considered separately.

Table 3.5: The number of susceptible and resistant reads used when training and evaluating the basic transformer model for different k-mers. Here, the resistant reads originate from subclass aac6p_class2 and the susceptible reads originate from dataset susceptible 1.

	Number of reads	
	S	R
Training	84626	3400
Validation	35750	1350

After the exploration of single k-mers we decided to extend the basic transformer model to be able to handle several different k-mers simultaneously as in the example in Figure 2.6. A combination of token appearances will probably improve the classifications since it provides the transformer with the opportunity to identify more dependencies in the genes. For this analysis we used resistant reads from subclass aph6 and thus susceptible reads from dataset susceptible 2. The reason for changing datasets once again, was that we wanted to see if the performance decreased significantly when one of the challenging gene classes was considered. Table 3.6 shows the sizes of the training and validation datasets.

Table 3.6: The number of susceptible and resistant reads used when training and evaluating the basic transformer model for combinations of k-mers. Here, the resistant reads originate from the challenging subclass aph6 and the susceptible reads originate from dataset susceptible 2.

	Number of reads	
	S	R
Training	18616	2950
Validation	7950	1200

If we use two different k-mers at the same the transformer model gets twice as big, and if we use three different k-mers at the same time the transformer model gets thrice as big. Therefore, we thought that the parameter value of the depth should be reconsidered. The idea was that if we increase the model by using combinations of k-mers, the depth can be decreased and thus also the computational time without affecting the performance. Which combinations of k-mers to consider were determined based on the result from the investigation of different k-mers. The combinations of interest became 2-mers and 3-mers, 4-mers and 5-mers and lastly 3-mers, 4-mers and 5-mers. Two transformer models with 2-mers and 3-mers but with different values of the depth were trained and evaluated. The depth value implying the best performance was then used when analyzing the combinations of 4-mers and 5-mers and also 3-mers, 4-mers and 5-mers.

3.6 Predictions of real metagenomic data

When the optimal parameter values and combinations of k-mers are used in the basic transformer model we call it our *final basic transformer model*. Now, we wanted to see how the final basic transformer model made predictions of the reads from the real metagenomic dataset. The final basic transformer model was therefore trained and evaluated using the data in Table 3.7. Here, reads are from the resistance datasets in Table 3.2 and from all available susceptible datasets. We wanted to use a big dataset with large diversity to make it easier for the transformer model to capture dependencies in the input reads. As a last step, we used this model to classify the one million reads in the real metagenomic dataset.

Table 3.7: The number of susceptible and resistant reads used for training and evaluating the final basic transformer model.

	Number of reads	
	S	R
Training	313242	32648
Validation	133700	13700

3.7 Performance metrics

To evaluate the performance of the transformer models where reads belong to either the susceptible class or the resistance class, we looked at how the reads in the validation data were predicted and compared the predictions with the true class labels. Figure 3.2 visualizes the performance in a confusion matrix. The resistant reads that were classified correctly as resistant are true positives, TP, and the susceptible reads that were classified correctly as susceptible are true negatives, TN. The resistant reads that were predicted as susceptible reads are false negatives, FN, and the susceptible reads that were predicted as resistant reads are false positives, FP.

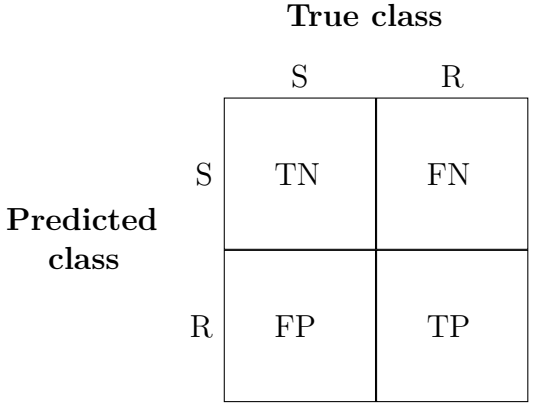

Figure 3.2: Confusion matrix for the two classes S and R. The confusion matrix was used to evaluate the performance of transformer models by calculating sensitivity and specificity.

We used the confusion matrix to calculate the sensitivity as

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP+FN}}, \tag{3.1}$$

and the specificity as

$$\text{specificity} = \frac{\text{TN}}{\text{TN+FP}}. \tag{3.2}$$

For evaluation, sensitivity and specificity are the most relevant performance metrics since the aim is to find all resistance reads without classifying any susceptible reads as resistant. Of course we wanted to get high specificity but it was even more important to get high sensitivity. Mostly, the transformer performance was evaluated by calculating the sensitivity and specificity at the last epoch. However, in some scenarios the sensitivity and specificity varied significantly over the last epochs. In such cases the mean and standard deviation of the sensitivity and specificity over the last five epochs were estimated.

So far when making classifications we have looked at the values of the output predictions for the susceptible and resistance class produced from the log-softmax function and assigned each read to the class with the highest score. In terms of probabilities, this is equal to classifying each read to the class which has a probability higher than 0.5. In some cases, 0.5 may not be the threshold implying the best performance, but it is possible to consider other thresholds as well.

By looking at different thresholds receiver operating characteristic curves, more known as ROC curves, can be created. To create a ROC curve in this project, we considered either probabilities with thresholds t between zero and one, or log-probabilities with thresholds t between the minimal and maximal value of the log-probabilities for the specific dataset. The advantage of the latter is that we get values spread out in a wider range which can be preferable if many values are approximately the same. However, in both cases the classifications were made as follows: If the value for S $< t$, then we classified the read as R and otherwise we classify the read as S. We did this for all values of t and calculated the corresponding specificity and sensitivity. A summary of the sensitivity and specificity for each threshold can then be visualized in a ROC curve. Figure 3.3 illustrates an example of a ROC curve where probabilities and ten equally distributed thresholds from zero to one have been considered to construct the blue ROC curve. The x-axis shows 1-specificity while the y-axis shows the sensitivity. Preferably, the ROC curve should be as close to the upper left corner as possible, because then both the values of sensitivity and specificity are high. In the figure, we also see a black dotted line which represents the case where reads are classified randomly.

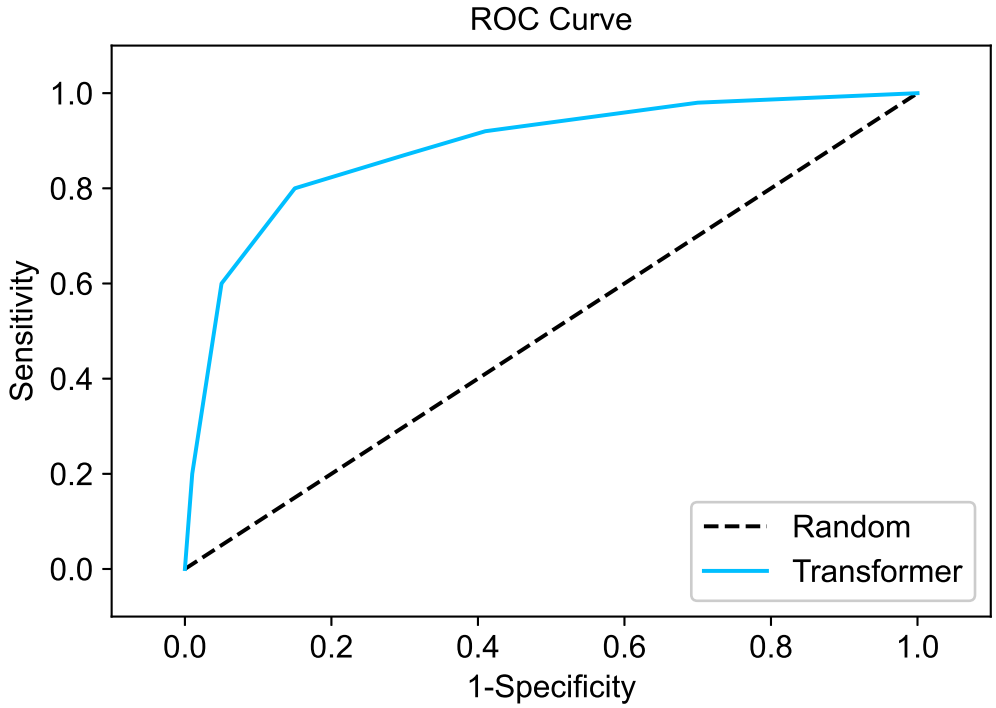

Figure 3.3: An example of a ROC curve illustrated in blue, where the x-axis represents 1-specificity and the y-axis represents sensitivity. The black dotted line represents the case where reads are classified randomly.

3.8 Multi-class transformer model

In addition to exploring and evaluating how well a transformer model can distinguish between a susceptible and resistance class, we also wanted to investigate if it is possible for the transformer model to distinguish between different resistance gene classes. The structure of the genes in the resistance gene classes should be more similar to each other than to the genes in the susceptible class. Hence, this should be a harder classification problem.

To construct the so-called *multi-class transformer model* we used the final basic transformer model with the same architecture, parameter values and k-mers. The only difference between the two models was that we now needed to extend the vocabulary with more class labels. Obviously, the number of class labels is the same as the number of classes in our input data. As output we now got a score from the log-softmax function for each class. Still, we assigned each read to the class with the highest score.

One option would be to implement a multi-class transformer model with nine class labels, one label for the susceptible class and one label for each resistance gene class in Table 3.2. However, for some of the resistance gene subclasses the available number of genes was rather small. For instance, the dataset aac3_class1 only contains

eight genes which most likely would make it hard for the transformer model to detect dependencies in the structure for this subclass. This is the reason why we determined to combine aac3_class1 and aac3_class2 into dataset aac3, and aac6p_class1, aac6p_class2 and aac6p_class3 into dataset aac6p. Thus, we ended up with the five resistance gene classes aac2p, aac3, aac6p, aph3p and aph6, which we wanted the multi-class transformer model to be able to distinguish between. Each resistance gene class was labeled and these labels were fed into the transformer model together with class label S which still represented the susceptible class.

The multi-class transformer model was trained and evaluated on the number of reads shown in Table 3.8 and in the table we also see how many reads were present for each class. The susceptible reads were simulated from dataset susceptible 1 and susceptible 2. Notice that the number of reads in the classes are different which may influence the performance. For instance, we expect that it will be hardest for the transformer model to classify reads from aac2p correctly since only 1000 reads are present in the training dataset.

Table 3.8: The number of reads used when training and evaluating the multi-class transformer model.

	Number of reads					
	S	aac2p	aac3	aac6p	aph3p	aph6
Training	103242	1000	7350	9350	11998	2950
Validation	43700	300	2800	4100	5300	1200

3.8.1 Performance metrics

Even if the multi-class transformer model makes predictions of six classes instead of two, the performance can still be summarized in a confusion matrix. Figure 3.4 is one example of such a confusion matrix. The performance of the model was now estimated using a one against all approach, meaning that we considered one class at a time. TP, TN, FP and FN are different depending on which class is considered. In the confusion matrix below aac2p is the class in focus. Then, TP is all reads from aac2p that were classified correctly, TN is all reads not belonging to aac2p that were not classified as aac2p, FP is all reads not belonging aac2p that were classified as aac2p and lastly, FN is all reads from aac2p that were classified incorrect. Sensitivity and specificity were now calculated for aac2p according to equations (3.1) and (3.2). When aac2p had been considered we moved on to the next resistance gene class and repeated the procedure. In the end, we got values of sensitivity and specificity for each resistance gene class.

		True class					
		S	aac2p	aac3	aac6p	aph3p	aph6
Predicted class	S	TN	FN	TN	TN	TN	TN
	aac2p	FP	TP	FP	FP	FP	FP
	aac3	TN	FN	TN	TN	TN	TN
	aac6p	TN	FN	TN	TN	TN	TN
	aph3p	TN	FN	TN	TN	TN	TN
	aph6	TN	FN	TN	TN	TN	TN

Figure 3.4: Confusion matrix for the multi-class transformer model with one susceptible class and five resistance gene classes. Sensitivity and specificity are calculated for the resistance classes by focusing on one resistance gene class at a time. The confusion matrix shows TN, FP, FN and TP if class aac2p is in focus.

Moreover, ROC curves can still be constructed even if multiple classes are present. The approach is now a two-step procedure. In the first step, we combined reads from all resistance gene classes into one resistance class and we classified reads as S or R in the same way as when only two classes were present. By doing so, we saw if the multi-class transformer model can distinguish between susceptible and resistance genes. In the next step, we considered one resistance gene class at a time and we assumed that all classes except for the considered one belonged to the same class. Then we did the classification of the reads for different thresholds in the same way as always. Now, we got ROC curves showing the performance of each resistance gene class.

4
Results

In this section we present the results obtained. First of all, we show in tables and figures how the sensitivity and specificity of validation data vary depending on different parameter values, k-mers and combinations of k-mers. For this investigation the basic transformer model was considered. Then, we present the results of both the final basic transformer model and the multi-class transformer model using the optimal values from the parameter and k-mer exploration. Besides presenting how the validation data is classified, we also show how the one million metagenomic reads from the real dataset are predicted using the final basic transformer model trained on an extended dataset with a bigger and more diverse set of susceptible genes.

4.1 Evaluation of parameter values

Table 4.1 summarizes the performance of the basic transformer model with 1-mers and varying values of the embedding size, number of heads, depth and batch size. Here, the general resistance genes are considered. The mean and standard deviation of sensitivity and specificity are based on the values of the validation data over the last five epochs.

Table 4.1: Mean and standard deviation of sensitivity and specificity of the last five epochs for different parameter values. For the standard setting the sensitivity mean is 0.620 with a standard deviation of 0.0412 and the specificity mean is 0.499 with a standard deviation of 0.0280. The sensitivity mean increases with a bigger value of the depth and decreases with a smaller embedding size or a smaller number of heads.

	Emb. size	**Number of heads**	**Depth**	**Batch size**	**Sensitivity** Mean	Std	**Specificity** Mean	Std
Standard	16	2	3	32	0.620	0.0412	0.499	0.0280
Change 1	8				0.594	0.0522	0.521	0.0381
Change 2		4			0.594	0.0863	0.526	0.0663
Change 3			6		0.670	0.0401	0.466	0.0254
Change 4				64	0.614	0.0352	0.503	0.0223

Evaluating the model with the standard settings, we notice from the table that the sensitivity mean becomes 0.620 with a standard deviation of 0.0412 and the mean specificity becomes 0.499 with a standard deviation of 0.0280. If the embedding size is decreased to 8 or if the number of heads is increased to 4, the sensitivity mean decreases to 0.594 while the specificity mean increases to slightly above 0.520. In the third change, the depth is doubled to 6 which implies that the sensitivity mean goes up to 0.670 but the specificity goes down to 0.466. Lastly, the model performance is more or less the same no matter if the batch size is 32 or 64.

We conclude from Table 4.1 that in order to reach the highest sensitivity the embedding size should be 16 and the model should have two heads and a depth of 6. The batch size in the basic transformer model does not affect the result. However, the advantage with increasing the batch size from 32 to 64 is that the computational time for one epoch decreases from 80 to 50 seconds. Therefore, embedding size 16, two heads, a depth of 6 and batch size 64 are used when exploring and evaluating different k-mers and combinations of k-mers.

4.2 Evaluation of *k*-mers

How the basic transformer model performs for different k-mers and the optimal parameter values obtained in Section 4.1 is shown in Table 4.2. Still, the mean and standard deviation of sensitivity and specificity for the validation data are estimated over the last five epochs. As mentioned earlier, now the resistant reads are from the resistance gene subclass aac6p_class2. From the table we see that the sensitivity mean is the lowest using 1-mers and the highest using 3-mers. However, for the transformer model where either 2-mers, 3-mers, 4-mers or 5-mers is used the sensitivity mean is at least 0.900 and the standard deviation is smaller than 0.0310. For 6-mers the sensitivity mean is 0.821. Moreover, we notice that the specificity mean increases with the value of k.

Table 4.2: Mean and standard deviation of sensitivity and specificity of the last five epochs for different k-mers. The sensitivity mean is never less than 0.900 with a standard deviation smaller than 0.0310 for the transformer models with 2-mers, 3-mers, 4-mers or 5-mers. The specificity mean on the other hand increases with an increased value of k.

	Sensitivity		**Specificity**	
	Mean	Std	Mean	Std
1-mers	0.727	0.0638	0.796	0.0426
2-mers	0.907	0.0197	0.897	0.0103
3-mers	0.959	0.0193	0.933	0.00610
4-mers	0.900	0.0309	0.972	0.00962
5-mers	0.921	0.0246	0.980	0.00600
6-mers	0.821	0.0299	0.992	0.00385

Based on the above stated result we considered it of interest to explore and evaluate if and how different combinations of 2-mers, 3-mers, 4-mers and 5-mers in the basic transformer model influence the result. Figure 4.1 contains four ROC curves which illustrate the performance of the basic transformer model with combinations of k-mers and resistant reads from subclass aph6. On the x-axis we have 1-specificity and on the y-axis we have sensitivity. The ROC curves are created using probabilities and 1000 equally distributed thresholds between zero and one. In the figure, both the orange and blue curve correspond to a combination of 2-mers and 3-mers but with the difference that the transformer model yielding the blue curve uses a depth of 3 and the orange curve uses a depth of 6. As seen, there is no clear difference in performance between these models. Anyhow, the computational time for the transformer model with depth 6 is around 70 seconds while it is only around 40 seconds with depth 3. Therefore, a depth of 3 is being used in the rest of the models in Figure 4.1 and also in the rest of the models in this master's thesis.

Looking at Figure 4.1 we also see that the purple ROC curve, which is a combination of 4-mers and 5-mers, implies an almost perfect result since both the sensitivity and specificity are close to one for certain thresholds. The green curve which is a combination of 3-mers, 4-mers and 5-mers imply an even better performance. Due to this, we choose to have a combination of 3-mers, 4-mers and 5-mers in our final basic transformer model and also in the multi-class transformer model.

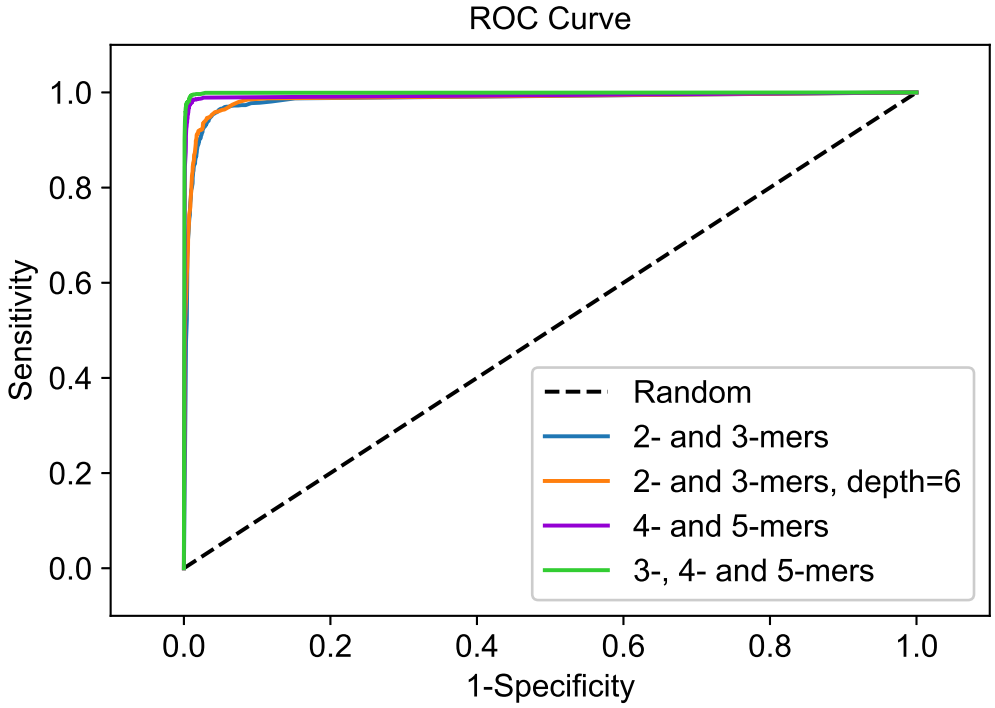

Figure 4.1: ROC curves for different combinations of k-mers based on probabilities and 1000 equally distributed thresholds between zero and one. The x-axis represents 1-specificity while the y-axis represents sensitivity. A combination of 3-mers, 4-mers and 5-mers in the basic transformer model gives the highest performance.

4.3 Final basic transformer model

The final basic transformer model is constructed using the previously found optimal parameters values and combinations of k-mers. More precisely, the settings are summarized in Table 4.3.

Table 4.3: Chosen parameter values and k-mers in the final basic transformer model.

Parameter	Value
Embedding size	16
Number of heads	2
Depth	3
Batch size	64
k-mers	3-mers, 4-mers and 5-mers

4.3.1 Performance on validation data

In the confusion matrix in Figure 4.2 we see exactly how many true general resistant reads and susceptible reads in the validation dataset that are predicted correctly in the final basic transformer model. We observe that 133521 susceptible reads and 13548 resistant reads are predicted correctly. The number of false positives is 179 and the number of false negatives is 152. Hence, to a great extent this model can distinguish between susceptible and resistance genes. Moreover, the transformer model takes about four hours to train and evaluate on the considered datasets.

		True class	
		S	R
Predicted class	S	133521	152
	R	179	13548

Figure 4.2: Confusion matrix for the validation dataset in the final basic transformer model. Most of the true susceptible reads are predicted as susceptible and most of the true resistant reads are predicted as resistant.

Using the values from the confusion matrix we get

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP+FN}} = \frac{13548}{13548 + 152} \approx 0.989$$

and

$$\text{specificity} = \frac{\text{TN}}{\text{TN+FP}} = \frac{133521}{133521 + 179} \approx 0.999.$$

Figure 4.3 also shows the performance on the validation dataset by visualizing the ROC curve of the transformer model with 1-specificity on the x-axis and sensitivity on the y-axis. The blue ROC curve is built up using probabilities and the values 0, 0.1, 0.2, ..., 1 as thresholds. Both sensitivity and specificity are close to one for all thresholds greater than zero but smaller than one. This indicates that the true resistant reads often are classified as resistant with high probability and that the true susceptible reads often are classified as susceptible with high probability.

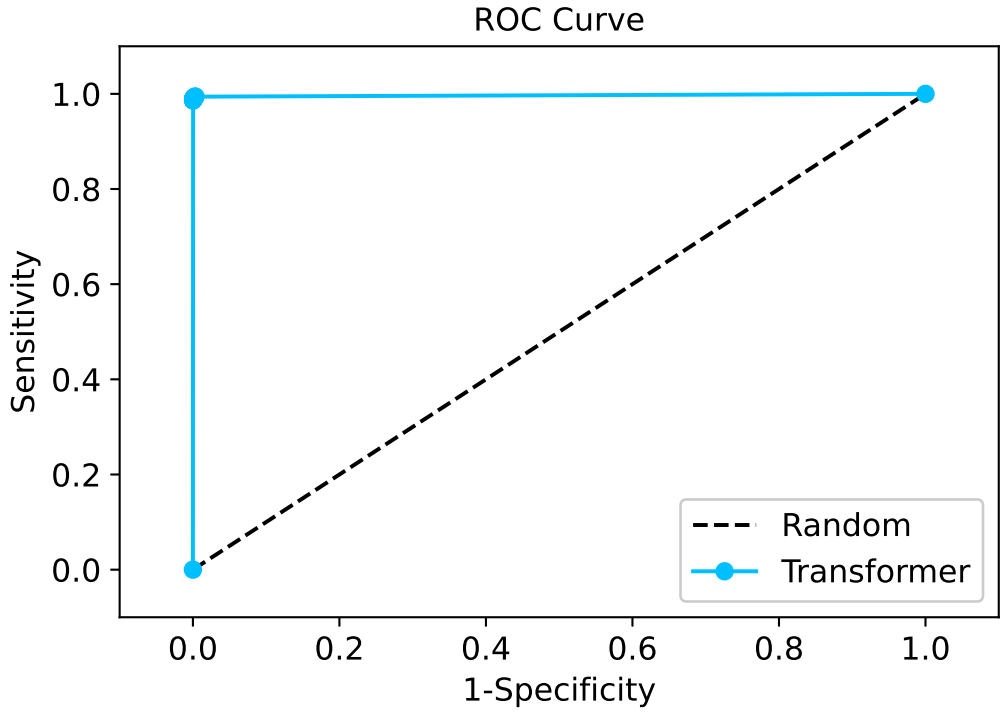

Figure 4.3: The ROC curve in blue illustrates the performance of the final basic transformer model. On the x-axis we have 1-specificity and on the y-axis we have sensitivity. As seen, the performance is nearly perfect.

4.3.2 Predictions of real metagenomic data

When feeding one million metagenomic reads from the real dataset to the trained and evaluated final basic transformer model in Section 4.3.1, it only takes a few minutes to make predictions of the reads. The transformer predicts 2371 of the reads as resistant. If we convert it to percent we get $2371/1000000 = 0.002371 = 0.2371\%$ predicted resistant reads. Many of these reads have in common that they seem to be very GC-rich and it is so far not clear how many of these reads that come from true resistance genes.

4.4 Multi-class transformer model

The parameter settings in Table 4.3 were used in the multi-class transformer model as well. In the confusion matrix in Figure 4.4 it is possible to see exactly how the reads in the validation dataset are predicted. For class S, aac6p, aph3p and aph6 most of the reads are predicted correctly. Reads from aac2p are often incorrectly classified as aac3. Approximately, half of the number of reads from aac3 are correctly classified.

		True class					
		S	aac2p	aac3	aac6p	aph3p	aph6
Predicted class	S	42973	3	149	78	34	40
	aac2p	40	79	80	41	28	9
	aac3	29	111	1527	92	412	27
	aac6p	274	11	394	3259	354	89
	aph3p	321	47	426	584	4359	77
	aph6	63	49	224	46	113	958

Figure 4.4: Confusion matrix for the validation dataset in the multi-class transformer model. Many reads from S, aac6p, aph3p and aph6 are classified correctly.

From the confusion matrix we estimate the sensitivity and specificity for each resistance gene class and we get the values stated in Table 4.4. The values confirm our previous observations. More precisely, aac2p is the class with the lowest sensitivity, only 0.263. The sensitivity for aac3 is 0.545. For aac6p, aph3p and aph6 the sensitivity is around 0.800 which is significantly higher. For all classes the specificity is above 0.970. Altogether, the multi-class transformer model performs best for aac6p, aph3p and aph6 according to these values.

Table 4.4: Sensitivity and specificity for the five resistance gene classes. The lowest sensitivity is 0.263 and is obtained for aac2p, while the highest sensitivity is 0.823 and is obtained for aph3p. The specificity is above 0.970 for all resistance gene classes.

	Sensitivity	**Specificity**
aac2p	0.263	0.997
aac3	0.545	0.988
aac6p	0.795	0.979
aph3p	0.823	0.972
aph6	0.798	0.991

In addition, Figure 4.5 shows how well the multi-class transformer model can distinguish between the resistance gene classes aac2p, aac3, aac6p, aph3p and aph6 when different thresholds are used. The figure shows ROC curves for all resistance gene classes separately and together. By together we mean how well the transformer model can distinguish between susceptible and resistant reads. Here, the ROC curves are based on the log-probabilities from the log-softmax function and 1000 thresholds. On the x-axis we have 1-specificity and on the y-axis we have sensitivity. From the figure we see that the multi-class transformer model to a great extent can identify reads belonging to aph6. For aac6p and aph3p the sensitivity and specificity are above 0.900 for some thresholds. The subclasses that are hardest for the transformer to identify are aac2p and aac3, but nevertheless the sensitivity and specificity are above 0.850 for some thresholds. Lastly, by looking at the brown ROC curve, we observe that it is easier for the transformer to discriminate between susceptible and resistance genes than between various resistance gene classes.

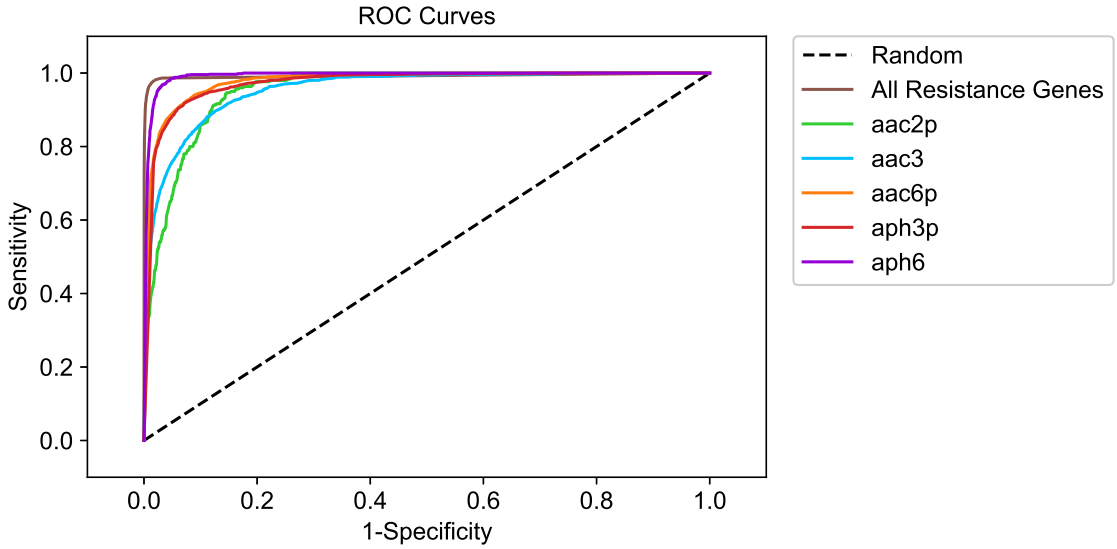

Figure 4.5: ROC curves for the different resistance gene classes separately and together. The x-axis represents 1-specificity while the y-axis represents sensitivity. Reads from class aph6 are easiest for the transformer to identify.

5
Discussion

Here, we analyze the results obtained in the previous section. We also make reflections, state some limitations and discuss reasons why the input data, parameter values and k-mers affect the predictions the way they do. As a last part, some examples of improvements of the transformer model are presented as future work.

5.1 The data

A limitation in this project was to only consider genes that were susceptible or resistant to aminoglycosides. This is a reasonable limit since it is unclear and had to be investigated if several antibiotic types can be considered simultaneously in a transformer model. Most likely, each type of antibiotic needs to be studied separately. If that is the case, the number of transformer models would have increased significantly. Moreover, a reasonable assumption is that the classification task is more or less the same no matter which antibiotic is considered. The optimal parameter values and combination of k-mers obtained for the transformer model with aminoglycosides should therefore be suitable even if the antibiotic changes. Due to this, it is of less importance to repeat the investigation for other antibiotics.

Throughout the project it has been observed that the amount of data and the DNA structure in the input datasets strongly reflects the predictions. For instance, patterns and dependencies that are typical for resistance genes need to be present in the datasets. Otherwise there is no possibility for the transformer model to classify reads from the resistance genes correctly. Also, if the number of reads is too small there is a chance that the transformer model finds patterns and dependencies in either the susceptible or resistance class that are actually not typical for the class, but happened to be present in the too small amount of data. Hence, a proper design of the input data is essential in order to achieve preferred classifications and predictions.

5.2 Parameter values

When we evaluated how different parameter values influenced the predictions we estimated the sensitivity and specificity by calculating the mean and standard deviation over the last five epochs. The reason for calculating means and standard deviations, instead of just focusing on the values at the last epoch, was that the results varied over the epochs. A higher sensitivity often implied a lower specificity

and vice versa. Perhaps, it would have been more convenient to rerun the model several times and then estimate the mean and standard deviation to get reliable results. However, since the aim was to just get a sense of which and how the parameters affected the results, we thought it was sufficient to analyze the results obtained from only one run.

In this project we limited the investigation of parameters to two values for each parameter. Therefore, we can not say for sure that the chosen values really are optimal. Another limitation was to only reconsider the value of the depth and not the other parameter values. The motivation was that the depth probably could be decreased when the model increased by considering multiple types of k-mers at the same time, which also was the case. Hence, we can not guarantee that the best values obtained when only 1-mers was considered also are the best once when other k-mers or combinations of k-mers are considered. On the other hand, there is nothing that indicates that other parameter values would improve the performance.

A parameter that was not investigated in this thesis was the sequence length of the input reads which was constantly 100 nucleotides. Longer reads would enable the transformer to recognize more patterns and dependencies and thus implying better performance with, most probably, the cost of more computational time. Due to this and the fact that we already got high performance we did not consider it of interest to investigate the sequence length. On the other hand, if a sequence length of 100 nucleotides had implied poor performance we would probably have increased it.

5.3 *k*-mers

After the optimal parameter values and combinations of k-mers had been found they were not reconsidered, but maybe the values and especially the optimal k-mers depend on the size of the datasets. Probably, there is a trade-off between the size of the dataset and the complexity of the vocabulary. The bigger the training dataset, the bigger k-mer can probably be used. If the k-mer is too big the training will be perfect but the validation will not be as good. For instance, if the size of the k-mer is equal to the length of the input reads, then the training would be perfect but the validation will not be as good. Because then the model learns dependencies in the training dataset that do not exist in the validation dataset.

The comparison of different k-mers in our project showed that our transformer model performs better for 2-mers, 3-mers, 4-mers or 5-mers than for 6-mers. At first sight, this behaviour was unexpected since more contextual information in the reads was then provided in the transformer model which would imply better performance. However, in our situation it seems like the complexity of the vocabulary may be too complex when 6-mers are used. Instead, it might be as described above, i.e. the transformer model finds patterns and learns dependencies in the training dataset that do not exist in the validation dataset. For example, the transformer model could gather information about the token AAGTCT and some other tokens,

but perhaps the combinations of these tokens do not exist in the validation dataset. The shorter the k-mer, the higher is the probability that the tokens and combinations of tokens present in the training dataset also are present in the validation dataset.

5.4 Final basic transformer model

Our final basic transformer model performed very well on the validation dataset. A majority of the reads were classified correctly with only a few amount of false negatives and false positives. This indicated that the structure of the reads in the validation dataset was similar to the structure of the reads in the training dataset. Using the same model, around 0.237 % of the reads in the real dataset were predicted as resistant and many of them were very GC-rich. Due to this, it is clear that the transformer does not just generate some random predictions of the reads. Instead, it has actually learned something based on what is in the training dataset. There is a possibility that our model considers GC-rich reads as resistant which might not always be true. Therefore, this set probably contains some false positives. Moreover, there is a chance that our set of susceptible predictions contains some false negatives since some patterns and dependencies might have been ignored by the transformer. If the design of our training dataset were different, the predictions would also be different.

5.5 Multi-class transformer model

From our results we noticed that when the multi-class transformer model was used the sensitivity was lowest for aac2p and aac3. The incorrectly classified reads from aac2p were probably generated due to the small amount of input data for the class. Probably, the transformer would have needed more data in order to find patterns and dependencies for aac2p. This may also explain why very few reads were predicted as aac2p overall. Not enough data could be the reason for the low sensitivity of aac3 as well. For aac6p, aph3p and aph6 the sensitivity was around 0.8 which was the highest among the resistance gene classes. Most likely, the sensitivity was higher for aac6p than for aac3 since more data was available for aac6p which made it possible for the transformer to find parts in the DNA structure that are typical for aac6p. As previously mentioned, the DNA structure of aph3p and aph6 are different compared to the other resistance gene classes. This made it easier for the transformer to separate these gene classes from the other even if the amount of data for aph6 was rather small.

5.6 Future work

Even if our transformer model performs very well when proper input data is used, there is still room for improvements. For example, it would be interesting to evaluate the transformer model on other antibiotics, not just aminoglycosides, and also

on several antibiotics simultaneously to see if the performance changes significantly. Most likely, the performance would be similar but on the other hand the DNA structure for different antibiotics varies which could both improve and worsen the performance of the transformer. It could also be of interest to investigate other parameter values and combinations of k-mers to see if the transformer model can be optimized further.

In this project, we made predictions of one million real metagenomic reads. The computational times were rather short and it only took some minutes to make predictions of this data. Therefore, it would be possible to perform a more extensive evaluation on real metagenomic data. It would be interesting to consider mote data since it would probably yield additional information about how the resistance genes look.

Another suggestion of future work is to explore how the input data should be designed depending on the prediction task. A question that could be taken into consideration is if the input data should be heterogeneous or homogeneous. Is it preferable to have as much diversity as possible? Also, it could be interesting to investigate how much data is needed for the transformer model to perform well.

6
Conclusion

An overall conclusion is that the transformer is a deep learning model which can be a useful tool for making classifications and predictions. Mainly, because it can detect dependencies between all parts in the input data no matter the distance. Also, exploring and evaluating different transformer architectures are helpful to achieve an optimal model suitable for the specific task. From this project we saw the importance of appropriate parameter values and combinations of k-mers in order to make classifications of metagenomic data with very good performance.

Based on the result from the final basic transformer model with real predictions and the multi-class transformer model with varying sizes of the resistance gene classes, we also conclude that the appearance and structure of the input data influence the performance significantly. Patterns and dependencies in a class must occur often enough so the model understands what the distinguishing features are. The exact design of the training and validation datasets depends on the classification task and should be further investigated to improve the performance of the transformer model.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[2] Centers for Disease Control and Prevention, "About Antibiotic Resistance." `https://www.cdc.gov/drugresistance/about.html`. [2021-02-15].

[3] K. Vong and K. Auclair, "Understanding and overcoming aminoglycoside resistance caused by n-6'-acetyltransferase," *MedChemComm*, vol. 3, no. 4, pp. 397–407, 2012.

[4] R. Ashman, "Genetic determination of susceptibility and resistance in the pathogenesis of Candida albicans infection," *FEMS Immunology & Medical Microbiology*, vol. 19, no. 3, pp. 183–189, 1997.

[5] L.-a. Pirofski and A. Casadevall, "Q&A: What is a pathogen? a question that begs the point," *BMC biology*, vol. 10, no. 1, pp. 1–3, 2012.

[6] F. Berglund, T. Österlund, F. Boulund, N. P. Marathe, D. J. Larsson, and E. Kristiansson, "Identification and reconstruction of novel antibiotic resistance genes from metagenomes," *Microbiome*, vol. 7, no. 1, p. 52, 2019.

[7] National Human Genome Research Institute, "Deoxyribonucleic Acid (DNA) Fact Sheet." `https://www.genome.gov/about-genomics/fact-sheets/Deoxyribonucleic-Acid-Fact-Sheet`. [2021-05-03].

[8] National Human Genome Research Institute, "A Brief Guide to Genomics." `https://www.genome.gov/about-genomics/fact-sheets/A-Brief-Guide-to-Genomics`. [2021-05-03].

[9] A. Grada and K. Weinbrecht, "Next-generation sequencing: methodology and application," *The Journal of investigative dermatology*, vol. 133, no. 8, p. e11, 2013.

[10] E. L. Van Dijk, H. Auger, Y. Jaszczyszyn, and C. Thermes, "Ten years of next-generation sequencing technology," *Trends in genetics*, vol. 30, no. 9, pp. 418–426, 2014.

[11] G. G. Chowdhury, "Natural language processing," *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[12] E. D. Liddy, "Natural language processing," 2001.

[13] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[14] Jaron Collis, "Glossary of Deep Learning: Word Embedding." `https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca`, 2017. [2021-05-19].

[15] Pytorch, "LOGSOFTMAX." `https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html`. [2021-05-20].

[16] Jay Alammar, "The Illustrated Transformer." `https://jalammar.github.io/illustrated-transformer/`, 2018. [2021-06-02].

[17] Lennart Svensson, "Transformers - Part 2 - Self attention complete equations." `https://www.youtube.com/watch?v=ER_KqqtoikA&t=201s`, 2020. [2021-05-20].

[18] Lennart Svensson, "Transformers - Part 3 - Encoder." `https://www.youtube.com/watch?v=SsLzwRXH0UI&t=353s`, 2020. [2021-05-20].

[19] K. Vong and K. Auclair, "Understanding and overcoming aminoglycoside resistance caused by n-6'-acetyltransferase," *MedChemComm*, vol. 3, no. 4, pp. 397–407, 2012.

[20] National Institute of Environmental Health Sciences, "ART." `https://www.niehs.nih.gov/research/resources/software/biostatistics/art/index.cfm`. [2021-02-16].

[21] Pytorch, "NLLLOSS." `https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html`. [2021-05-26].

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY