Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Detection of tail fiber proteins via machine learning methods

Bachelor Thesis

2021 Jozef Bača

Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Detection of tail fiber proteins via machine learning methods

Bachelor Thesis

Study program:	Applied Informatics
Field of Study:	Informatics
Department:	Department of Applied Informatics
Supervisor:	Andrej Baláž, MSc.

Bratislava, 2021

Jozef Bača

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jozef Bača
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Detection of tail fiber proteins via machine learning methods.
Detekcia proteínov chvostových vlákien pomocou metód strojového učenia.

Anotácia: Bakteriofágy sú vírusové organizmy, ktoré infikujú a ničia baktérie. Bakteriofágy obsahujú proteíny chvostového vlákna (tail fiber protein), ktoré majú potenciál pri diagnóze a následnej liečbe bakteriálnych infekcií. Identifikácia týchto proteínov je laboratórne náročná čo vytvára priestor pre informatický nástroj, ktorý by systematicky prehľadal dostupné verejné databázy, identifikoval spoločné vlastnosti chvostových vlákien a použil tieto spoločné vlastnosti na ich detekciu spomedzi skupiny proteínov s neznámou funkciou.

Cieľ: Cieľom práce je klasifikácia chvostových vlákien. Z pohľadu strojového učenia sa jedná o úlohu učenia s učiteľom, kde vstupy sú textové reťazce a výstupy sú priradenia do dvoch tried. Študent, použitím existujúcej databázy bakteriofágov a dostupných bioinformatických nástrojov, pripraví dataset proteínových sekvencií vo forme textových reťazcov. Tento dataset, využitím informácií z verejných databáz, rozdelí na sekvencie reprezentujúce proteíny chvostového vlákna, sekvencie s inou známou funkciou a sekvencie s neznámou funkciou. Následne na takto vyčistenom datasete skonštruuje klasifikátor a vyhodnotí jeho predikčnú schopnosť.

Vedúci: MSc. Andrej Baláž
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 02.10.2020

Dátum schválenia: 17.02.2021 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Acknowledgement: I am extremely grateful to my supervisor MSc. Andrej Baláž for his unwavering support throughout this work and invaluable insight into machine learning. With his motivation and patience, he gave me profound belief in my work. I must also thank doc. Mgr. Tomáš Vinař, PhD. for his insightful suggestions, which helped to improve my work. Thanks also go to Mgr. Jaroslav Budiš, PhD. and Mgr. Michal Kajsik, PhD. for their great leadership of this project and for providing me the opportunity to be part of it. Special thanks to my family for supporting me during my studies.

Abstract

Bacteriophages are virus organisms that infect and destroy bacteria. They are used in phage therapy, which has great potential for future diagnosis and treatment of bacterial infections. Phages use their tail fibers to identify and attach themselves to host bacteria. The goal of this thesis is to create a classifier, which can classify tail fiber proteins of phages. It will look at how the data are collected and prepared, what machine learning methods can be used and what their effectiveness is.

Keywords: bacteriophage, classification, machine learning, logistic regression, support vector machine

Abstrakt

Bakteriofágy sú vírusové organizmy, ktoré infikujú a ničia baktérie. Používajú sa v bakteriofágovej terapii, ktorá má veľký potenciál pre budúce diagnózy a liečbu bakteriálych infekcií. Bakteriofágy používajú svoje chvostové vlákna na identifikáciu hostiteľská baktérie a prichytenie sa na ňu. Cieľom tejto práce je vytvoriť klasifikátor, ktorý by vedel klasifikovať proteíny chovstových vlákien bakteriofágov. V práci sa pozrieme na zdroj dát a ich spracovanie, možné metódy strojového učenia a na ich efektívnosť.

Kľúčové slová: bakteriofágy, klasifikácia, strojové učenie, logistická regresia, metóda podporných vektorov

Contents

List of Figures

List of Tables

Introduction

Bacterial infections have been on the rise ever since the appearance of multidrug-resistant and extremely drug-resistant bacteria in the 1980s. With sparse prospects of producing new antibiotics and the list of effective ones shortened every year, scientists have once again returned to examine the potential of bacteriophages and their use as a powerful treatment [16] of mutating bacteria. Phages are one of the most abundant and the most diverse of life forms. It is estimated that there are over $4.8 \cdot 10^{31}$ phages on Earth [5] and are omnipresent in the soil, water and even in our food. Bacteriophages are viruses that infect and kill bacteria and bacteria only while having virtually no negative effect on humans or animals[20]. These properties make them the ideal candidate to introduce brand new methods to combat ever more resistant bacteria.

The medical use of bacteriophages is nothing out of ordinary, as the first studies in this field took place before the Second World War, but had been discontinued once antibiotics were invented. It was believed that bacteria could not develop resistance to antibiotics, which unfortunately led to overuse and the first cases of resistance have been reported as early as 1979.

In this work, we focus on identifying the tail fiber proteins. The identification of tail fibers via biological methods is expensive and time-consuming. While we have large databases of phage genomes, a substantial part of the genes has not yet been properly annotated with their corresponding function and around a quarterof the genes remain without any assigned function [8]. AI can help resolve this issue based on common properties among identified genes. The first part of this thesis presents needed biological background and already existing software. The second chapter shows the data and its preparation. The third section aims to show the implementation of machine learning methods and validate their accuracy. The last chapter summarizes results and findings.

Chapter 1

Methods

1.1 Definitions of the basic biological terms

1.1.1 Genome

A genome is a set of information in an organism's DNA [3]. Just like the DNA, it is commonly written in a four-letter nucleotide alphabet consisted of capital Roman characters G, C, A and T representing the nucleotides most frequently found in DNA. The information carried are instructions for all the proteins the organism will ever synthesize [3]. Each cell's DNA in our bodies consists of genomes and will pass them to both daughter cells when cell division occurs. The genome sequence of an organism gives us information about the properties and functions of the genome.

1.1.2 Gene

The genes are the basic physical unit of inheritance [10] and are arranged on structures called chromosomes. They consist of segments of DNA that contain instructions for building molecules, most of which are proteins [10]. These proteins then carry out the function [10]. The definition of what exactly is a gene has been lately questioned in the scientific community, but it is still a useful turn in this thesis.

1.1.3 Genome versus Gene

Genes are in length significantly shorter than Genomes, as the information of the latter carries instructions for even 30,000 proteins in some cases [3].

1.1.4 Protein

Proteins are complex molecules, that are linked in sequences to form a functional molecule [21]. They are made up of a large quantity of amino acids. The sequence of amino acids determines each protein's unique 3-dimensional structure and its specific function [25]. They serve as antibodies defecting our body from viruses and bacteria, or as the transmitters of signals which coordinate biological processes in our bodies. They carry out the most basic of functions in our bodies and, as a matter of fact, in phages too.

1.1.5 Phages and tail fibers

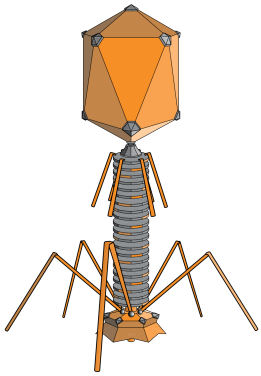


Figure 1.1: Bacteriophage is composed of a head with stored genetic material and tail, which is made of tail tube and tail fibers.

Bacteriophages (phages) are viruses that invade bacterial cells and kill the host bacteria. In the 1940s Soviet Union and the Eastern Block were developing phage therapy. The basis of this treatment is binding of phage to bacterial cells and causing rapid lysis of the cells [16], in practical terms, killing the bacteria without any harm to human cells. With the overuse of conventional drugs and the consequent rise of drug-resistant superbacteria, this treatment could become an effective treatment for bacterial illnesses. Although modern medicine and biology have made great advances towards identifying phage proteins, a segment needed to correctly use phages as a form of treatment, most phage proteins have not yet been identified. In this work, we attempt to tackle the problem of the identification of phage tail fibers with machine learning.

Phages, as shown in figure 1.1, consist of 3 main parts [12]:

1. tail - longer middle part of the body, which functions as protein infection promoting structure
2. capsid - resembles head situated on the top of the tail, contains the genetic material of the virus
3. tail fibers - receptor binding proteins that come in two sets: long tail fibers which can detect the right host bacteria and short tail fibers that trigger the infection process.

Tail fibers are essential for successful infection of a bacteria, as the phages use them to recognize the host and attach themselves to it.

1.2 Available tools

The first step in addressing the problem of phage tail fibers identification was the research of the current literature on the topic. We identified multiple studies with similar goals. Next, we describe their approaches and compare the differences in their methods.

1.2.1 VIRALpro

VIRALpro [8] was developed using Support Vector Machines (SVM) to detect and identify capsid and tail sequences. To refer to the tail fibers, it was fitted on a dataset of 4895 tail fiber sequences with 10-fold cross-validation on the training set, using one of the folds as a validation set.

The study claims that the F1-score on the validation set is 92,6%, although the size of the mentioned dataset is relatively small.

1.2.2 DeepCapTail

DeepCapTail [1] was designed to outperform VIRALpro. For this purpose, deep neural network models were used, for their known exceptional performance. K-mer frequencies were used as features to train the models, with k-mer sizes of 1 to 4.

Best F_1-score, when detecting tail fiber sequences, was achieved with a model structure consisting of 4 layers of nodes 600:300:150:60 and k-mer size of 2, reaching 0.93.

1.2.3 PhANNs

PhANNs [4] is an open-source machine learning tool that can classify proteins into one of ten classes and non-phage proteins into a class called "other". Tail fiber is one of those classes. Feature sets used are composition of 2-mers, 3-mers, 4-mers and side-chain groups. Its model is an Artificial Neural Network ensemble and was trained on features extracted from an 11-fold cross-validation and reached F_1-score of 0.875.
While overall accuracy of PhANNs is admirable 86.2%, tail fiber is among those classes where PhAANs struggles, with values of F_1-score on our validation dataset being 55,36%. PhANNs was published in April 2020 and is the latest tool we looked into. The study aimed to create a faster and more accurate tool than VIRALpro.

1.3 Machine learning

In this section, we go through machine learning models and methods used in this work. We also discuss the basics of machine learning itself and the main concepts associated with learning. The work of this thesis consisted of using classification models, thus this section discussed theory only related to classification.

1.3.1 Basic Framework

In our daily life, we are facing countless problems, which we try to define, analyze and then solve based on our analysis. The world is, however, not always analytically describable or easily assessable. Machine learning can be used to deal with problems, to which we have no analytical solutions. Instead, it helps us to construct an empirical solution, without any theory, regarding a given problem, based on the data describing it. The data is the determining element of machine learning. They form the basis for machine learning to be able to predict solution to the problem. Its values represent influential factors from the real world. These discrete units of information are called data points and they form the input for machine learning, e.g. how much one likes comedy. Based on them we want to get to the optimal unknown function $f : \mathbf{X} \to \mathbf{Y}$, where $\mathbf{X}$ is the input space for all possible input sets $\mathbf{x}$ and $\mathbf{Y}$ is the output space. Function f represents the optimal solution function for our problem and is unknown as it cannot be analytically defined, however, we know what its outputs are from the

data. The data is a data set of pairs of inputs and outputs $(x_1, y_1), (x_2, y_2), (x_3, y_3), ...$ with size n, where for $i = 1, 2, ..., n$:

$$y_i = f(x_i). \tag{1.1}$$

Each y_i is the correct decision for data point x_i. Function f is therefore the target function, to which we want to acquire a function as similar as possible.

$$\mathbf{y} = f(\mathbf{x}) \tag{1.2}$$

The algorithm chooses and compares functions from the set of hypothesis $\mathbf{H}$. In this work we will refer to the best hypothesis function as h. Input for this function will be the data input $\mathbf{x} \in \mathbf{X}$. In (1.1) we can substitute h for f and get for $i = 1, 2, ..., n$ with input size n:

$$y_i = h(x_i) \tag{1.3}$$

or from equation (1.2)

$$\mathbf{y} = h(\mathbf{x}). \tag{1.4}$$

The function $h(\mathbf{x})$ requires set of parameters $\mathbf{w}$, also known as weights, each corresponding to one input from the set $\mathbf{x}$. Parameters define what shape does the hypothesis function have. Finding the best set $\mathbf{w}$ is the main goal of machine learning models. They estimate parameters via their learning algorithms. The main challenge is to find the right hypothesis function h from the hypothesis set $\mathbf{H}$, which is infinite.([2], Components of Learning)

The h we are looking for is the function from $\mathbf{H}$ that has the smallest error rate $E(h)$. To calculate $E(h)$ let us first introduce function I:

$$I(y_i \neq h(x_i)) = \begin{cases} 0 & \text{if } y_i = h(x_i) \\ 1 & \text{if } y_i \neq h(x_i) \end{cases}. \tag{1.5}$$

To amplify the error rate $E(h)$ appropriately, we want the function I to return value only if misclassification happens, which in our case is when the result of function $h(\mathbf{x})$ differs from the expected value $\mathbf{y}$. Having defined I, we can write error rate formula:

$$E(h) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq h(x_i)) \tag{1.6}$$

where n is the size of input set. ([13], The Classification Settings)

1.3.2 Logistic regression

Logistic regression is a model of linear classification used to classify inputs into two classes. Therefore it separates inputs that are wanted from those that are not, which makes it appropriate for binary decisions, although another possible use is also making a probabilistic prediction. Typical examples include detecting spam e-mails and predicting heart disease. Positive and negative classes represent values 1 and 0 respectively, with values in the former being the ones that are accepted and the ones that are not in the latter. The cornerstone of logistic regression is calculating the prediction of input and that is the result of its hypothesis function h. Because probability prediction is always a number between 0 and 1, so is h.

$$0 \leq h(\mathbf{x}) \leq 1 \tag{1.7}$$

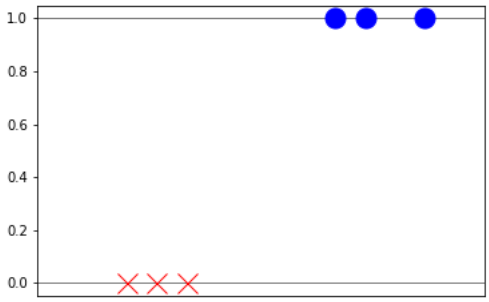


Figure 1.2: Data points: red Xs represent negative samples and blue circles represent positive samples.

To calculate the predicted value, we use equations, where the output depends on the chosen hypothesis θ and the input x.

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \tag{1.8}$$

This format returns a predicted value, which can be any number. In order to get the probabilistic value, we use the sigmoid function ($(sigm)$), which maps any real number to a value between 0 and 1.

$$sigm(x) = \frac{1}{1 + e^{-x}} \tag{1.9}$$

Then we can say that for logistic model with h

$$h(\mathbf{x}) = sigm(\mathbf{w}^T\mathbf{x}). \tag{1.10}$$

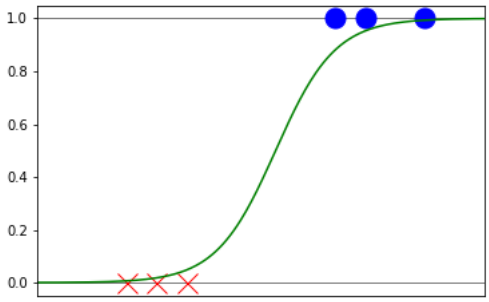


Figure 1.3: Sigmoid function graph.

Let us have some data points, positive and negative 1.2. After applying the logistic model using sigmoid function, we will be able to separate the data points by their value 1.3. The sigmoid function is referred to as a soft threshold and can help us determine an error measure for learning from the data. The data gives us sample probabilities and [2] therefore generate a distribution (noisy target) $P(y|\mathbf{x})$

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = 1 \\ 1 - h(\mathbf{x}) & \text{for } y = 0 \end{cases} \tag{1.11}$$

$P(y|\mathbf{x})$ is therefore equal to $h(\mathbf{x})$ or $1 - h(\mathbf{x})$, based on the value of y. The goal of learning is now to estimate parameter vector θ, which we can do by estimating the error of the current θ. Consider N data points labeled 0 or 1. Based on equation (1.11)

- for all data points labeled 1, we want to estimate $\mathbf{w}$ such that $P(y|\mathbf{x})$ is as large as possible (as close to 1 as possible), so maximize $\prod_{i=1}^{N} h(\mathbf{x}_i)$
- for all data points labeled 0, we want to estimate $\mathbf{w}$ such that $1 - P(y|\mathbf{x})$ is as large as possible (as close to 1 as possible), so maximize $\prod_{i=1}^{N}(1 - h(\mathbf{x}_i))$

Error measure of $\mathbf{w}$ is equal to the product of these 2 possibilities.

$$E(\mathbf{w}) = \prod_{i=1}^{N} P(y|\mathbf{x}) \tag{1.12}$$

By applying method of maximum likelihood on (1.12), the error measure $E(\mathbf{w})$ is equal to

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\Big(\frac{1}{sigm(y_n \mathbf{w}^T \mathbf{x}_n)}\Big). \tag{1.13}$$

The most common way to minimize this error is to use the gradient descent algorithm. Gradient descent is an iterative optimisation algorithm that finds the local minimum of a twice-differentiable function, such as $E(\mathbf{w})$. Error measure function $E(\mathbf{w})$ is always convex in the case of logistic regression, which consequently implies that found local minimum will also be the global minimum of the function. An important parameter of gradient descent is the step size η, which determines the gradient itself. When we take a step, we also need to know in which direction the step was taken. That is represented by direction vector $\mathbf{v}$. Therefore if we take a step of size η in the direction $\mathbf{v}$, the new weights are $\mathbf{w}(0) + \eta\mathbf{v}$. The gradient descent in logistic regression is iterative, so to calculate the gradient at the iterative step t, we use equation

$$\mathbf{g}_t = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{x}_n \theta^{\mathrm{T}}}} \tag{1.14}$$

where $\mathbf{g}_t$ is the gradient at step t. To optimalize the weights (minimalize the error E) in logistic regression we use the gradient descent in following steps [2]:

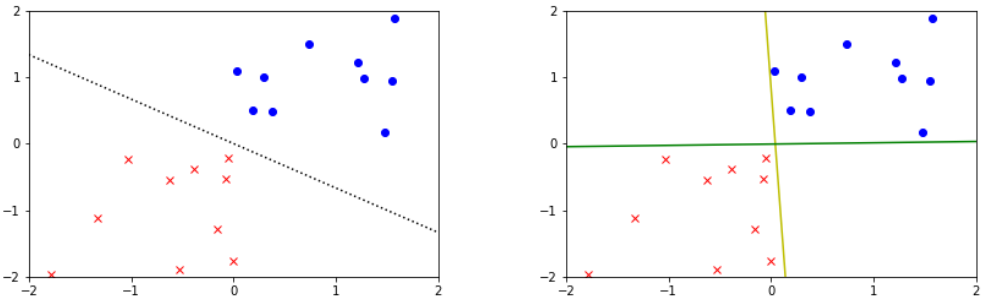
1. Initialize the weights $\mathbf{w}$, so that at step $t = 0$ the weight is $\mathbf{w}(0)$.
2. Set step t to 0.
3. Compute the gradient for step t using (1.14).
4. Set the direction $\mathbf{v}_t = -\mathbf{g}_t$.
5. Update weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\mathbf{v}_t$.
6. Iterate to $t = t + 1$ and return to step 3., unless it is time to stop.
7. Return computed weights $\mathbf{w}$.

Now, with this algorithm, we are able to choose the best hypothesis function $h(\mathbf{w})$, which will return the probability of its input being in a positive class. The last step is to set a threshold. All probabilities above it are classified as positive and all below as negative.

1.3.3 Support vector machine

The *support vector machine* (SVM) is a supervised-learning model that was first introduced in 1995 by Vladimir Vapnik, who at the time worked for AT&T Bell Labs. in the USA [7]. In this section, we explain the main ideas of SVM and show the basic mathematics behind it. All information was greatly influenced by Kilian Weinberger's lecture on SVM [15].

SVM is a supervised learning method that works on a similar basis as a perceptron, in that it is also an algorithm that will find a hyperplane. The main difference is that perceptron will find any hyperplane that divides the data points into two distinct regions, while SVM will find the hyperplane with the *maximal margin*. Another difference is that the target classes, negative and positive, are represented by values -1 and 1 respectively. Firstly we discuss how we define a hyperplane and what role does it play, what maximal margin is and how do we find it and finally what options do we have when data is not linearly separable.



(a) Hyperplane defined $1+2X_1+3X_2 = 0$. (b) 2 examples of hyperplanes.

Figure 1.4: Positive (blue circles) and negative (red xs) data points in 2-dimensional space. Graph 1.4a shows one example of a possible partition of space, for positive data points applies $1 + 2X_1 + 3X_2 > 0$ and for negative points $1 + 2X_1 + 3X_2 < 0$. Graph 1.4b shows example of hyperplanes which divide the space, but not optimally.

Let us have a p-dimensional space with a $p-1$-dimensional hyperplane. Let us also have a point X, that is on this hyperplane. The mathematical description of such hyperplane is [9]:

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0 \tag{1.15}$$

where $\beta_0, ..., \beta_p$ are hyperplane parameters and point $X = (X_1, ...X_p)^T$. Hyperplane in

(1.15) divides the p-dimensional space into two distinct regions, in which for any point X holds:

- $\beta_0 + \beta_1 X_1 + ... + \beta_p X_p < 0$ if point X is on one side
- $\beta_0 + \beta_1 X_1 + ... + \beta_p X_p > 0$ if point X is on the other

All data points from the training set with p features can be represented as points in p-dimensional space. The main idea is then to find such a hyperplane, that divides this space into 2 regions. One with only positive points and the other with only negative points. An example of this division is shown in figure 1.4a. This hyperplane would be the result of a perceptron. The problems arise, however, when perceptron finds a hyperplane that is closer to one group of data points than to the other, which is not an optimal solution, with examples shown in 1.4b.

While perceptron finds any hyperplane that divides the positive and negative points, SVM will find hyperplane, which "maximizes the distance to the closest data points from both classes"[15]. This distance is called *maximal margin*. Separating hyperplane with maximal margin is the one farthest from the training data points, which makes it the optimal separating hyperplane [9]. To find maximal margin, we must find the shortest distance from the separating hyperplane to each training data point. Minimal shortest distance from them is called *margin* γ. The maximum margin hyperplane is the one hyperplane, for which the margin is the largest. The maximal margin hyperplane leaves the maximal possible space between itself and the training points, which we believe, provides enough space to avoid incorrectly classified data points when classifying the test dataset.

Hyperplane is a set of points defined with parameters $\mathbf{w}$ (which also is a perpendicular vector to $\mathcal{H}$), b for some point $\mathbf{x}$ on the hyperplane by

$$\mathcal{H} = \{\mathbf{x} | \mathbf{w}^T \mathbf{x} + b = 0\}. \tag{1.16}$$

Let us take some point $\mathbf{x}$ from the training dataset. Let $\mathbf{x}^P$ be the projected point $\mathbf{x}$ onto $\mathcal{H}$, so $\mathbf{x}^P \in \mathcal{H}$. Then there exists vector $\mathbf{d}$ such that $\mathbf{d} = \mathbf{x} - \mathbf{x}^P$, which means that norm of $\mathbf{d}$ represents the distance between point $\mathbf{x}$ and $\mathcal{H}$, and also that $\mathbf{d}$ is perpendicular to $\mathcal{H}$. Vector $\mathbf{d}$ is then parallel to $\mathbf{w}$, therefore $\mathbf{w}$ can be rescaled to $\mathbf{d}$ by some scale α: $\mathbf{d} = \alpha \mathbf{w}$. Taking into account $\mathbf{x}^P \in \mathcal{H}$, we can substitute (1.16):

$$\mathbf{w}^T \mathbf{x}^P + b = \mathbf{w}^T (\mathbf{x} - \mathbf{d}) + b = \mathbf{w}^T (\mathbf{x} - \alpha \mathbf{w}) + b = 0$$

which implies

$$\alpha = \frac{\mathbf{w}^T\mathbf{x} + b}{\mathbf{w}^T\mathbf{w}}.$$

To get the shortest distance, we compute the Euclidean norm of $\mathbf{d}$:

$$\|\mathbf{d}\|_2 = \sqrt{\mathbf{d}^T\mathbf{d}} = \sqrt{\alpha^2\mathbf{w}^T\mathbf{w}} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\sqrt{\mathbf{w}^T\mathbf{w}}} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|_2} \tag{1.17}$$

Then the margin γ (minimal shortest distance) with trainig dataset D and hyperplane with $\mathbf{w}, b$ is defined by

$$\gamma(\mathbf{w}, b) = \min_{\mathbf{x}\in D} \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|_2}. \tag{1.18}$$

Now we want to find such $\mathbf{w}$ that maximizes the margin:

$$\max_{\mathbf{w},b} \gamma(\mathbf{w}, b) \text{ such that } \forall i\, y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 0. \tag{1.19}$$

which using (1.18) can be formulated as

$$\max_{\mathbf{w},b} \frac{1}{\mathbf{w}^T\mathbf{w}} \min_{\mathbf{x}\in D} |\mathbf{w}^T\mathbf{x} + b| \tag{1.20}$$

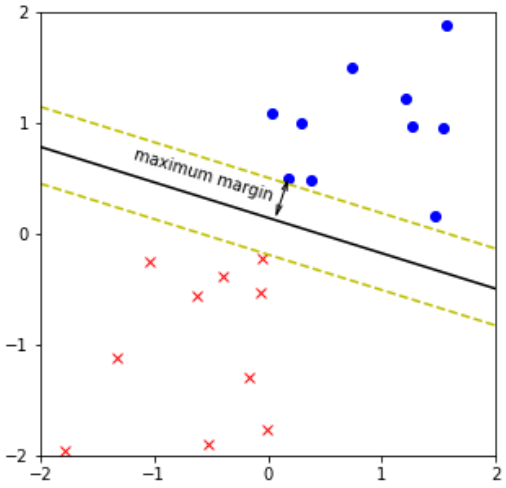


Figure 1.5: Maximum margin hyperplane graph.

To find maximum of a minimum requires complex calculations and so we need to simplify this equation. We can do that thanks to the definition of a hyperplane $\mathcal{H} = \{\mathbf{x}|\mathbf{w}^T\mathbf{x} + b = 0\}$, where we can multiply $\mathbf{w}$ and b by any constant and it would

still remain the same hyperplane, we would only rescale the vector $\mathbf{w}$. Let us then fix the scale of $\mathbf{w}$ such that

$$\min_{\mathbf{x}\in D} |\mathbf{w}^T\mathbf{x} + b| = 1. \tag{1.21}$$

Substituting this into (1.20) we get

$$\max_{\mathbf{w},b} \frac{1}{\mathbf{w}^T\mathbf{w}} \cdot 1 = \min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w}. \tag{1.22}$$

The optimization problem then has to consider 3 constraints from (1.19, 1.21, 1.22):

1. $\forall i\, y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 0$
2. $\min_{\mathbf{x}\in D} |\mathbf{w}^T\mathbf{x} + b| = 1$
3. $\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w}$

We can join the constraints 1. and 2. and get 2 constraints

1. $\forall i\, y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$
2. $\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w}$.

When the optimal $\mathbf{w}$ and b are found, that means that some data points in the training dataset fit the optimal constraint. For some i-th data point $\mathbf{x}$ from the training dataset the constraint would be

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1. \tag{1.23}$$

We say that these data points have *tight* constraint and we call these data points *support vectors*. These are the points, that are the closet to the hyperplane and their distance to the hyperplane is the maximal margin.

With data from the real world it is often the case, that the positive and negative points from the training dataset cannot be separated by a hyperplane, therefore no support vectors with no maximal margin would be found, in other words, no data points would satisfy the tight constraints. However, there is also another type of constraint - *soft* constraint. It allows the optimization problem to ignore some data points which violate the tight constraint. Firstly let us introduce variable ξ_i, which permits $\mathbf{x}_i$ to be closer to the hyperplane than the maximal margin. The soft constraint is

1. $\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i$
2. $\forall i\, y_i(\mathbf{w}^T\mathbf{x} + b) \geq 1 - \xi_i$
3. $\forall i\, \xi_i \geq 0$

where parameter C controls the strength of the ξ parameter. Another way to use SVM on linearly inseperable data is to use *kernel functions*.

1.3.4 Kernel functions and kernel trick

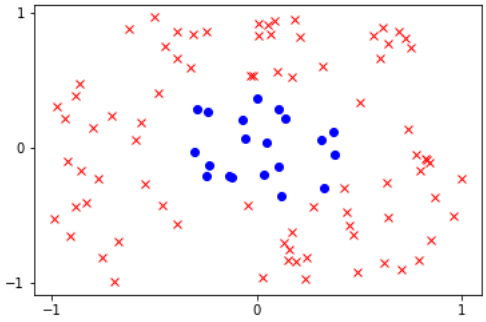


Figure 1.6: Non-linearly separable data. Blue dots represent positive and red xs represent data points.

Kernel functions are used by SVMs to extract features from the training data. They also work as a similarity function between 2 data points. For two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ from the training data, the the kernel function K is defined as the dot product:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle. \tag{1.24}$$

where ϕ is a mapping function which we will discuss later. For linear classification, SVMs use linear kernel function [14]:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \tag{1.25}$$

Many models, like Support Vector Machine or Logistic regression, rely on the data being linearly separable. However, in the real world, this is rarely the case and we need to transform the data in such a way, that we achieve linear data separability.

Kernel trick is a function that maps the input space to a feature space. To achieve linear data separability, it transforms the input space into a higher dimensional feature space, so that the data is then linearly separable in this higher dimensional space. Let us consider mapping function ϕ from 2-dimensional input space to 3-dimensional feature space:

$$\phi((x, y)) = (x, y, x^2 + y^2) \tag{1.26}$$

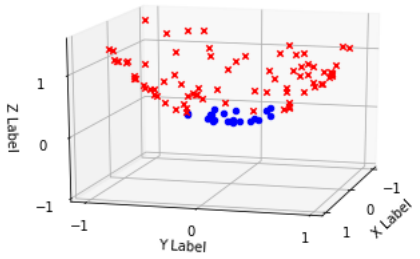


Figure 1.7: Non-linear separable data from Figure 1.6 mapped by function 1.26.

We can now notice that a non-linearly separable data set as shown in Figure 1.6 has transformed into a linearly separable data set shown in Figure 1.7, thus making previously inapplicable linear classification models suitable for this data set [11]. For example a kernel function using ϕ from (1.26) would take form

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j + \|\mathbf{x}_i\|^2 \|\mathbf{x}_j\|^2 . \tag{1.27}$$

There are many kernel functions, among most used are the polynomial kernel, Radial Basis Function and Sigmoid Kernel.

1.3.5 Radial basis kernel

Radial basis kernel, also known as Gaussian Kernel or Radial Basis Function (RBF), is a kernel function designed for training datasets that are not linearly separable. It computes the Euclidean distance to figure out how far away from each other are data points from the same class. For data points $\mathbf{x}_i$ and $\mathbf{x}_j$ from the training set, RBF is defined as [9]:

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \sum_{k=1}^{p} (\mathbf{x}_{ik} - \mathbf{x}_{jk})^2) \tag{1.28}$$

where *gamma* γ is some positive constant. In (1.28) the bigger the distance $\sum_{k=1}^{p} (\mathbf{x}_{ik} - \mathbf{x}_{jk})^2$ is, the smaller result $exp(-\gamma \sum_{k=1}^{p} (\mathbf{x}_{ik} - \mathbf{x}_{jk})^2)$ becomes. When we look at the kernel function as a measure of similarity, the bigger the distance is, the less similar 2 data points are. Similarity is also indirectly proportional to how large gamma γ is, therefore it defines how much influence does a single training data point have [24]. The larger gamma γ is, the closer the other data point has to be.

In this work, we will use RBF on our training data and analyze results with different values of gamma γ and C parameter of the soft constraint.

1.3.6 String kernel

Until now, we have discussed p-dimensional data points $\mathbf{x}$ with p features that were given by the training dataset. These features can be some precomputed k-mers that the machine learning models do not do anything about. However, it can be beneficial to merge the feature extraction with the model fitting process. This has been mostly used when the training data are texts, strings of characters that can represent messages, voice interactions, or biological information as protein or DNA strings. The feature extraction is then done by the kernel function and is called *string kernel*.

String kernels are functions that use the occurrence of the same substrings as the features to obtain the similarity properties of 2 strings. String kernels can vary in their function and so we will now describe the string kernel we fitted on our training dataset in this work. We chose an algorithm, that compares strings' substrings of length $k = 2$. The steps of the algorithm are:

1. create dictionary d for each string s, where keys are all possible substrings of length k and values are the number of its occurrences
2. iterate over all pairs of strings get rid of all substrings that are not common in at least one pair, let us call the final set of substrings U.
3. iterate over all pairs and compute their similarity with the function K from (1.29)

$$K(s_i, s_j) = \sum_{u \in U} s_i[u] \cdot s_j[u] \tag{1.29}$$

Let us demonstrate on a example, where set of strings $S = \{\text{mommy}, \text{mummy}, \text{mum}, \text{ummm}\}$. First we calculate all possible substrings of length $k = 2$ shown in Figure 1.1.

Next we take those substrings that are common with at least 2 strings and get their number of occurrences shown in Figure 1.2.

With (1.29) we can show the similarity between strings mommy and mummy as $K(\text{mommy}, \text{mummy}) = 1 \cdot 1 + 1 \cdot 1 = 2$. The final kernel is shown in figure 1.3.

string	**substrings**
mommy	mo, om, mm, my
mummy	mu, um, mm, my
mum	mu, um
ummm	um, mm

Table 1.1: Strings from S with their substring possibilities of length $k = 2$. We can notice that substrings that are common for at least 2 strings $U = \{\text{mm}, \text{my}, \text{mu}, \text{um}\}$.

string \ substring	mm	my	mu	um
mommy	1	1	0	0
mummy	1	1	1	1
mum	0	0	1	1
ummm	2	0	0	1

Table 1.2: Strings from S with number of occurrences of substrings from U.

string \ string	mommy	mummy	mum	ummm
mommy	2	2	0	2
mummy	2	4	2	3
mum	0	2	2	1
ummm	2	3	1	5

Table 1.3: Resulted kernel using our string kernel function with strings S.

1.3.7 Datasets

Before we can use any machine learning models, we need to prepare our data for the models. Important fact to keep in mind is that we cannot use all of the data to fit the models, as the model is biased towards any data it was based on. Firstly we need to create a dataset that does not influence the model and is only used once the model tweaking is finished to test the model. That is why we call it *test dataset*. The rest of the data is used to create the model. With most of data we train the model, we call

them the *training dataset* and the remaining data is used to analyze the performance of our models to tweak its parameters. This set of data is called *validation dataset*.

As for the ratios of these datasets, a rule of thumb says that 20% of the data should be assigned to the validation set [2]. In practise, it can be smaller depending on how many parameters the model has. Likewise, the test dataset should also be around 20% and the remaining 60% should be the training dataset.

1.3.8 Possible problems in data

We will now present a few problems that can occur within the data itself.

An imbalanced dataset means that there are more positive data points in the dataset than negative ones. The model is then more influenced by one class data points than the other. It can also mean that the model does not have enough data points of one class to determine the difference between them. One possible solution is to randomly delete some data points from the more populated class so that the sizes are comparable. A similar but more complicated solution is to generate more data points of the underrepresented class based on the ones you already have.

Feature extraction is a strategically important moment in machine learning, as it determines the values that represent the data. The more features we have, the easier it can be for the model to classify data points, but more features also mean that more computational power is required. K-mers are a good example of this problem. The larger k we choose, the more features we have that represent the data which can lead to more precise classification. It however also means, that with 20 letters that are used to describe a protein genome and $k = 2$, we have $20^2 = 400$ possible features. With $k = 3$, bigger just by 1, there are $20^3 = 8000$ possible features.

As we demonstrated in section 1.3.6, kernel functions can get quite complex. The final kernel that needs to be calculated has a size of $N \times N$ for N being the size of the training dataset. Each cell of this kernel is computed with kernel functions for each pair of data points. These are some of the problems that we encountered during our work.

Chapter 2

Data and bioinformatics tools

In this chapter, we will present the source of our data and bioinformatics tools we used to prepare the dataset. All components were free and publicly available at the time this work was made.

2.1 Data

2.1.1 Phage database

In our work, we decided to use the virus database of the National Center for Biotechnology Information (NCBI). NCBI's database has been widely used in numerous publications on virus genomes and sequences [17]. Our dataset is comprised of data from NCBI's database [18] downloaded on the 31st of August 2020. This data mainly consisted of the phage references, their functions and genomic DNA sequences. DNA sequences were needed to be annotated to fully represent genomic features. For this part, the Prokka tool 2.2.1 was used and the resulting dataset included fully annotated genomic features that were written as strings of 20 capital letters of the alphabet. It was these strings that we utilized as data representations of phage proteins.

2.1.2 K-mer features

When the data in machine learning is presented as strings, one important question arises: how to get the features that the models would use from the strings. This problem can be sometimes addressed in the model itself, e.g. using string kernels in

SVMs shown in 1.3.6, but not all models can implement such functions. One of the ways to represent strings as data is to take substrings of length k and count their occurrence in the string. Let us take the word BANANA and let us try a few different values of k (shown in figure 2.1).

k	substrings
1	B, A, N, A, N, A
2	BA, AN, NA, AN, NA
3	BAN, ANA, NAN, ANA
4	BANA, ANAN, NANA

Table 2.1: K-mers of BANANA with different values of k.

Let us follow this by counting the number of occurrences of each substring.

substring	A	B	N
value	3	1	2

(a) $k = 1$

substring	AN	BA	NA
value	2	1	2

(b) $k = 2$

substring	ANA	BAN	NAN
value	2	1	1

(c) $k = 3$

substring	ANAN	BANA	NANA
value	1	1	1

(d) $k = 4$

Table 2.2: Tables show number of occurrences of substrings from figure 2.1. Using substrings as features, one can represented words with numbers.

The final result is that we can compare how similar words based on how many instances of the same substrings are in the words. Used substrings of length k as features are called k-mers. Finally we are only left dealing with the parameter k and what value should it hold. This can only be done by testing different values of k, which requires extensive computational power.

2.2 Tools

Listed tools are bioinformatics tools that we needed to process the biological data into forms that machine learning models can work with.

2.2.1 Prokka

Genome annotation is the process of identifying and labeling all the relevant features on a genome sequence [22]. Genome annotation is required when working with protein genome data. Most tools created were server or mail-based, but in 2014 new installable command-line tool was introduced called Prokka [23]. Unlike its predecessors which provided results in hours, Prokka can annotate a typical bacterial genome in around 10 minutes on an average desktop computer. For this reason, it is a great option when a larger dataset of genomes needs to be annotated. Prokka's annotation process has 2 steps:

1. Prokka finds coordinates of genes in a genome sequence (f.e. finds gene starting on 20th position in the sequence and has length 100)
2. Prokka tries to assign a function to every gene by comparing each one of them to databases of known proteins - UniProt, RefSeq, Pfam and TIGRFAMs

2.2.2 Biopython

Biopython is a freely available open-source tool based on *python* programming language. It is a cluster of modules intended to simplify computational biology and bioinformatics scripts and software [6]. It includes module Bio.SeqIO, which makes reading and writing bioinformatics format FASTA easier for programmers and enhances software performances. Together with *numpy* and *matplotlib* offers great foundation for bioinformatics programming.

2.2.3 Sci-kit

As the demand for statistical analysis and machine learning methods grew, *Sci-kit learn* was created to be an intuitive and user-friendly python package integrating the state-of-the-art algorithms [19]. Written mostly in *python* programming language, it can be easily incorporated in software with other most commonly used scientific modules, especially with *numpy*. Most notably, it includes models for linear regression, logistic regression and support vector machine.

Chapter 3

Implementation

The goal of this work was to try different machine learning models to detect phage tail fiber proteins and analyse models' performances. In the present chapter, we will go through the creation of our datasets, implementations of models and present their validation. Validations will be shown using confusion matrix 3.1 and its derivations (F1 score, TPR - True Positive Rate, FPR - False Positive Rate, TNR - True Negative Rate, FNR - False Negative Rate).

Predicted / Actual	tail fiber	other protein
tail fiber	True Positives	False Negatives
other protein	False Positives	True Negatives

Table 3.1: Illustration of confusion matrix.

3.1 Preparation of the dataset

As the first step, we had to prepare our datasets. This started with downloading the data from an existing database, followed by using bioinformatics tools to acquire protein sequences, computing their features and ended with creating the training, validation and test datasets.

After we downloaded data from the NCBI's database (see 2.1.1), we used Prokka (see 2.2.1) to annotate the genomes. After that, we omitted hypothetical proteins.

Those are proteins whose function is yet unknown. Using proteins with known functions, we created the fundamental dataset, which consisted of 54,627 protein sequences. Out of them, only 5,602 were tail fibers. With sequences prepared, we transformed the dataset to contain features instead of the whole sequences. K-mer features with $k = 2$ were chosen because they were relatively easy to compute. With these features, the dataset contained 400 features.

54,627		
training dataset	valdiation dataset	test dataset
34,961	8,740	10,926

Table 3.2: Sizes of final datasets.

We then randomly divided this dataset into training, validation and test datasets. Instead of using common ratios of 60:20:20, we decided to approximately use ratios 65:15:20 as we did not intend to tweak a lot of parameters in machine learning models. Division is shown in figure 3.2. At that point, we had our datasets prepared and were ready to start training models.

3.2 Logistic regression

Logistic regression is one of the most basic and easiest classifying machine learning models and therefore was a good choice to start with. We used scikit's model `linear_model.LogisticRegression` to fit the training dataset.

Actual \ Predicted	tail fiber	other protein
tail fiber	680	195
other protein	103	7762

Table 3.3: Confusion matrix of logistic regression model on validation dataset.

We proceeded with predicting the validation dataset. The resulted confusion matrix is shown in table 3.3 and confusion matrix rates in figure 3.1. Without any tweaking, the linear model achieved a true positive rate of 77,71% and an F1-score of 82,03%.

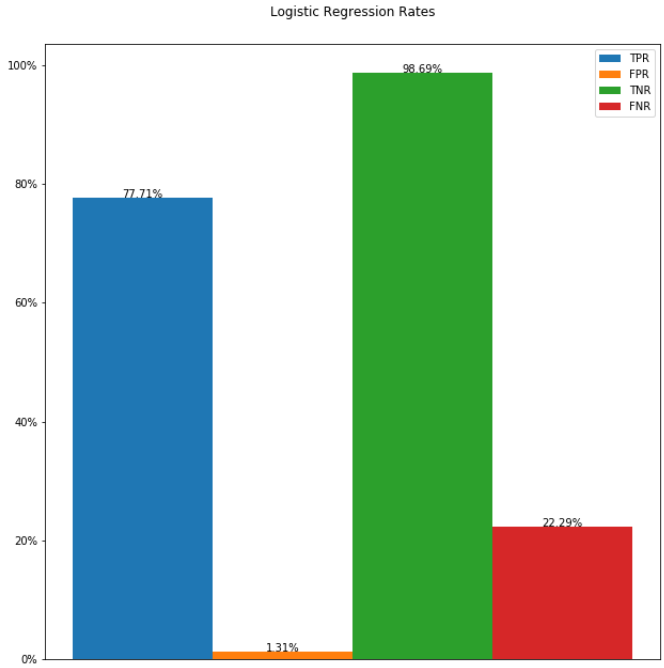


Figure 3.1: Graph shows values of rates of the confusion matrix.

At 98,69% the true negative rate shows the influence of an unbalanced dataset with only 10% of sequences being tail fibers. However, that did not indicate, that the model was confident with classifying the negative class, because the false negative rate was at a relatively high 22,29%, compared to false positive rate at 1,31%. The accuracy of a binary classifier can also be illustrated with ROC curve. It is acquired by plotting true and negative positive rates, that are achieved with different threshold values of the model. ROC curve of this model is shown in figure 3.2.

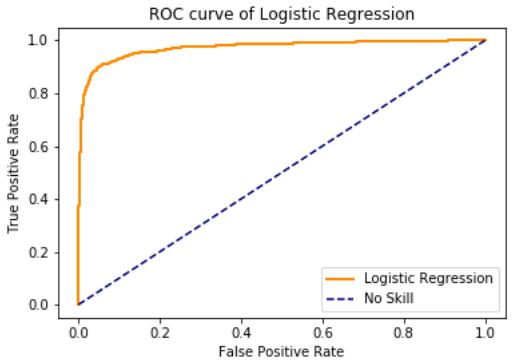


Figure 3.2: ROC curve of our logistic regression model.

3.3 Logistic regression with chosen features

Having trained the logistic regression model, we can extract from it the importance of each feature. *Feature importance* tells us, how much influence does every feature have on the classification process. To speed up the classification, we can then choose only those features, that have a strong influence on it, lowering the number of features that need to be computed. Using scikit's `clf._coef`, we chose the 40 most influencing

Actual \ Predicted	tail fiber	other protein
tail fiber	163	712
other protein	88	7777

Table 3.4: Confusion matrix of logistic regression model trained on chosen features with the highest influence.

features of the first model and fitted another logistic regression model with the dataset restricted to only those features. Confusion matrix of this model is shown in table 3.4. In the rates graph 3.3 it is clearly visible, that this model is not as accurate as the previous one. With F1-score of only 28,95% and true positive rate of 18,63%, this model is not an improvement.

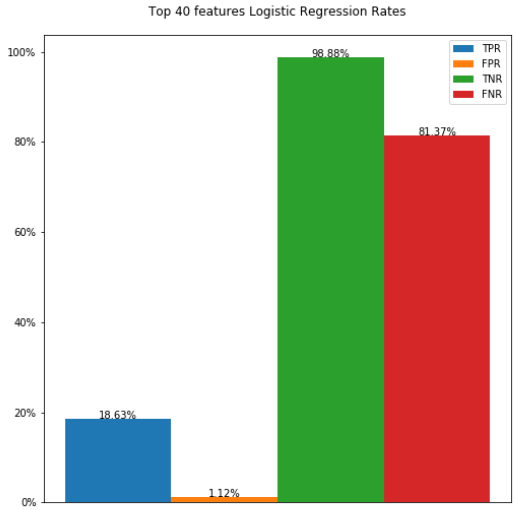


Figure 3.3: Graph shows values of rates of confusion matrix of logistic regression model with 40 chosen features.

3.4 Linear support vector machine

Support vector machine was our second choice to fit the training dataset. SVMs are more computationally demanding, but are usually more accurate than the previously fitted models. We implemented scikit's model `svm.SVC()` without changing any of the default values of its parameters, therefore the model used a linear kernel function. Confusion matrix of this model is shown in table 3.5. This model had great F1-score

Actual \ Predicted	tail fiber	other protein
tail fiber	769	106
other protein	11	7854

Table 3.5: Confusion matrix of support vector machine model on validation dataset.

of 92,93%. True confusion matrix rates 3.4 are fairly high, while false positive and negative rates are at only 0.14% and 12,11% respectively.

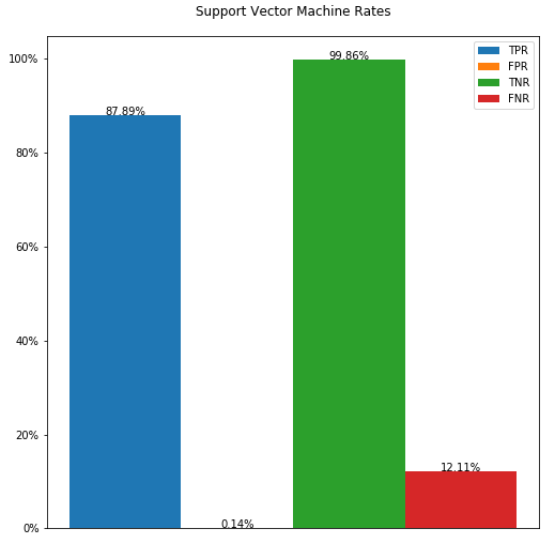


Figure 3.4: Graph shows values of rates of confusion matrix of a simple support vector machine model.

3.5 Support Vector Machine with RBF kernel

We followed the linear SVM model by fitting a SVM with RBF kernel function. Additionally, we wanted to try different values of parameters C and gamma γ. To get a first glance at the impact of these parameters on model accuracy, we chose combinations of values $0.1, 1, 10$ for C and $0.01, 1, 100$ for gamma γ. In graph 3.5, we can see the

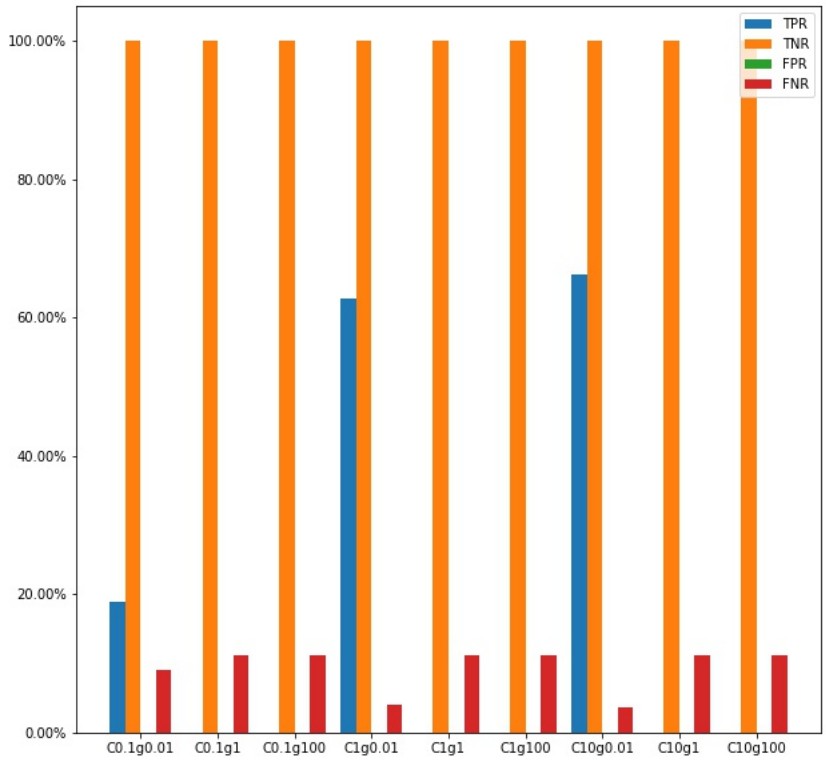


Figure 3.5: Graph shows confusion matrix rates for svm models using rbf kernel function with values 0.1, 1, 10 for C and 0.01, 1, 100 for gamma γ.

confusion matrix rates for these models. Most noticeable is the fact, that all models had almost 100% true negative rates. Just as visible is the notion, that only models with gamma $\gamma = 0.01$ had true positive rates higher than 0%. That lead us to believe, that in future models, C parameter is not so influential and gamma γ should be at most 0.01. And so we fitted 3 further models, all with $C = 1$ and values of gamma γ $0.001, 0.0001$ and 0.00001.

Graph 3.6 shows confusion matrix rates of these 3 models. All of them outperform

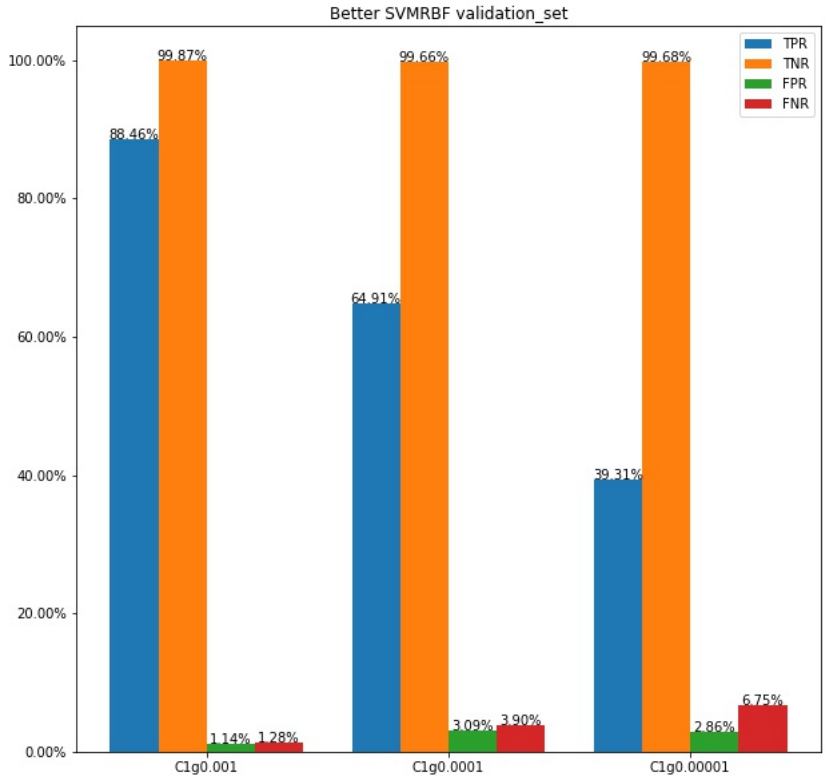


Figure 3.6: Confusion matrix rates of SVM RBF models with values $C = 1$ and gamma γ $0.001, 0.0001$ and 0.00001.

models with previous values of parameters. A model with gamma $\gamma = 0.001$ was the best out of the 3 with the highest true positive and true negative rates. The F1-score of this model was 93,31%. Its confusion matrix is shown in table 3.6.

Actual \ Predicted	tail fiber	other protein
tail fiber	774	101
other protein	10	7855

Table 3.6: Confusion matrix of support vector machine model using radial basis function with $C = 1$ and gamma $\gamma = 0.001$ on validation dataset.

3.6 Best model

The last part of our work was to identify the best model based on these validations. We decided to choose the one with the best F1-score.

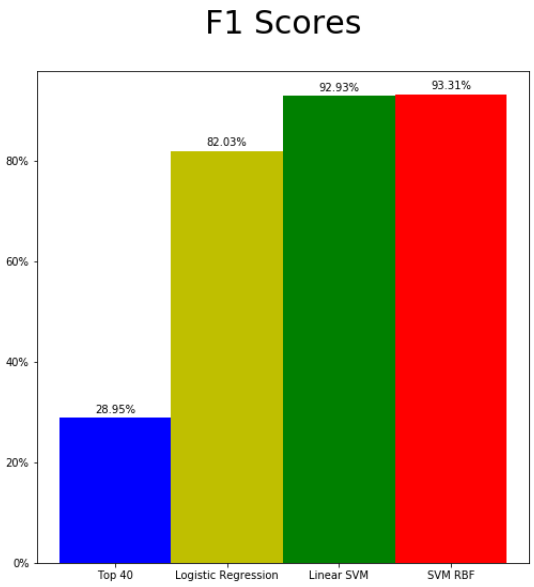


Figure 3.7: Graph shows F1 scores of our models achieved on validation dataset.

The best one was the support vector machine using radial basis function with parameters $C = 1$ and gamma $\gamma = 0.001$. We then validated this final model on the test dataset. Confusion matrix of this test dataset validation is shown in table 3.7. The final model achieved an F1-score of 94,7% on the test dataset.

Actual \ Predicted	tail fiber	other protein
tail fiber	1027	109
other protein	6	9784

Table 3.7: Confusion matrix of support vector machine model using radial basis function with $C = 1$ and gamma $\gamma = 0.001$ on test dataset.

3.7 PhANNs comparison

From the tools we looked into, PhAANs was the most modern. We, therefore, wanted to compare its performance to our best model. The performance of PhAANs on the test dataset is shown in table 3.8.

Actual \ Predicted	tail fiber	other protein
tail fiber	863	273
other protein	1065	8725

Table 3.8: Confusion matrix of PhANNs on test dataset.

On this dataset, PhANNs reached an F1-score of 56,33%. Our best model from previous section 3.6 reached F1-score of 94,7% on the test dataset.

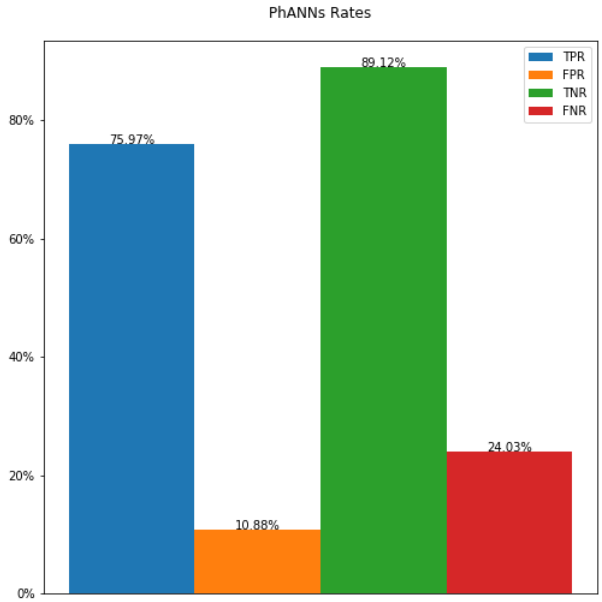


Figure 3.8: Confusion matrix rates of PhANNs on test dataset.

Chapter 4

Discussion

In this work, we trained several models to classify protein genome sequences into the positive and negative categories. Based upon conducted research and our accomplishments, in this chapter, we discuss remarks for future works in a similar field.

Throughout our work, we only considered k-mers features of length 2, as this was the least computationally demanding option of k. Higher k values would provide the models more divergent features to work with, helping to describe the similarity of two strings in more detail. Another convenient kernel function when dealing with strings is the string kernel. SVM using string kernel computes the strings' features by computing its kernel, unlike linear or RBF kernels, which need the features precomputed in the training dataset. Less computation means faster model classification performance.

The logistic regression model showed us, how the unbalanced dataset influenced the model's decision-making process and what sort of confusion matrix rates can be achieved. It also provided us with information on feature importance. However, when we tried to capitalize on this and create a model focused on these features, its performance was far worst than before. This prompted the idea, that feature importance cannot be obtained from models, but rather should be founded on biological properties. It could also be the case, that there are no particular 2-mer features that would decide the class and that all features are somewhat significant.

Support vector machine surprised us with remarkable accuracy even without making any tweaks to its parameters, though once we started tweaking them, the performance grew better. Parameter values we used were roughly estimated to give us an idea of how they influence the model's performance. It was clearly demonstrated, that SVM

with well-chosen kernel functions and parameter values can be very precise. Instead of approximately estimating them, advanced data science methods could find better values than the ones we used.

Conclusion

The goal of this work was to use machine learning models to classify tail fiber proteins of bacteriophages, which could replace the current demanding lab work in the future. This meant to process the existing data, try different machine learning models and validate their performances.

In the first chapter, we explored the biological background and terms surrounding bacteriophages, explained the importance of tail fibers and their genomes to phage therapy. We looked into already existing tools such as VIRALpro and DeepCapTail and investigated their functionality. Later we also validated tool PhANNs and compared its F1-score to our model. We explored machine learning hypothesis, introduced logistic regression and support vector machines, and researched their mathematical functions and error calculations.

In the second section, we went through the data acquiring process. We presented NCBI's database and downloaded its data. After that, we described the k-mer feature extraction and gave examples of its use. Here we introduced the bioinformatics tools Prokka and Biopython, which enabled us to work with data and prepare the datasets. We also mentioned the sci-kit package, whose models we used for machine learning.

The third chapter contains the details and figures we obtained during our implementation. Firstly we computed the 2-mer features and explained the ratios we used to divide the data into training, validation and test datasets. We displayed the use of logistic regression on the training dataset as an easy classification model that can show us the feature importance. We then explored the application of support vector machine models. Linear kernel implementation already showed better performance than logistic regression without any tweak to the default parameter values. We then trained more models with RBF kernel changing the C and gamma γ values, which revealed the importance of gamma γ parameter to get better accuracy from the SVM model. Based

upon this we trained a model with $C = 1$ and gamma $\gamma = 0.001$ that accomplished the best performance among our models with an F1-score of 94,7%. We then compared this model to the existing PhANNs tool to get an idea, of how this model behaves in contrast to existing tools.

In the last chapter, we discussed the option of better performance with higher k value k-mer features, at the expense of more demanding computation. We also summarised the models' performances and gave suggestions for future research in this field.

To sum up, we used an existing NCBI phage database to explore data with bioinformatics tools. We then created a dataset with extracted features and divided it into datasets, that were used in the machine learning process. Machine learning was done using logistic regression and support vector machine models and we analysed their performances using figures and confusion matrices.

Bibliography

[1] Dhoha Abid and Liqing Zhang. "DeepCapTail: A Deep Learning Framework to Predict Capsid and Tail Proteins of Phage Genomes". In: *bioRxiv* (2018). DOI: 10.1101/477885. eprint: https://www.biorxiv.org/content/early/2018/11/23/477885.full.pdf. URL: https://www.biorxiv.org/content/early/2018/11/23/477885.

[2] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. Vol. 4. AMLBook, 2012.

[3] B. Alberts et al. *Molecular Biology of the Cell 4th Edition*. Routledge, 2002. ISBN: 9780815332886. URL: https://www.ncbi.nlm.nih.gov/books/NBK26821/.

[4] Vito Adrian Cantu et al. "PhANNs, a fast and accurate tool and web server to classify phage structural proteins". In: *PLoS computational biology* 16.11 (2020), e1007845.

[5] Ana Georgina Cobián Güemes et al. "Viruses as winners in the game of life". In: *Annual Review of Virology* 3 (2016), pp. 197–214.

[6] Peter JA Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (2009), pp. 1422–1423.

[7] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297. URL: https://link.springer.com/content/pdf/10.1007/bf00994018.pdf.

[8] Clovis Galiez et al. "VIRALpro: a tool to identify viral capsid and tail sequences". In: *Bioinformatics* 32.9 (Jan. 2016), pp. 1405–1407. eprint: https://academic.oup.com/bioinformatics/article-pdf/32/9/1405/24343069/btv727.pdf. URL: https://doi.org/10.1093/bioinformatics/btv727.

[9] James Gareth et al. *An Introduction to Statistical Learning with Applications in R*. Springer Publishing Company, Incorporated, 2014.

[10] *Gene*. National Human Genome Research Institute. 2020. URL: https://www.genome.gov/genetics-glossary/Gene (visited on 04/12/2021).

[11] Martin Hofmann. "Support vector machines-kernels and the kernel trick". In: *Notes* 26.3 (2006), pp. 1–16. URL: https://cogsys.uni-bamberg.de/teaching/ss07/hs_rc/slides/SVM_Seminarbericht_Hofmann.pdf.

[12] Paul Hyman and Timothy Harrah. "Tail fiber function and structure". In: *Bacteriophage T4 Tail Fibers as a Basis for Structured Assemblies*. ASME Press, Jan. 2014. ISBN: 9780791860373. DOI: 10.1115/1.860373_ch2. URL: https://doi.org/10.1115/1.860373%5C_ch2.

[13] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.

[14] *Machine Learning for Intelligent Systems, Lecture 13: Kernels*. Conrell University. 2018. URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote13.html (visited on 05/22/2021).

[15] *Machine Learning for Intelligent Systems, Lecture 9: SVM*. Conrell University. 2018. URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html (visited on 05/22/2021).

[16] Majid Taati Moghadam et al. "How phages overcome the challenges of drug resistant bacteria in clinical infections". In: *Infection and drug resistance* 13 (2020), p. 45.

[17] *Our Data Model*. National Center for Biotechnology Information. 2020. URL: https://www.ncbi.nlm.nih.gov/labs/virus/vssi/docs/about/#our-data-model (visited on 05/25/2021).

[18] *Our Data Model*. National Center for Biotechnology Information. 2020. URL: https://www.ncbi.nlm.nih.gov/labs/virus/vssi/#/virus?SeqType_s=Nucleotide&VirusLineage_ss=Bacteriophage,%5C%20all%5C%20taxids&Completeness_s=complete&Proviral_s=exclude&EnvSample_s=exclude (visited on 05/25/2021).

[19] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[20] Nicola Principi, Ettore Silvestri, and Susanna Esposito. "Advantages and limitations of bacteriophages for the treatment of bacterial infections". In: *Frontiers in pharmacology* 10 (2019), p. 513.

[21] *Protein*. National Human Genome Research Institute. 2020. URL: https://www.genome.gov/genetics-glossary/Protein (visited on 04/13/2021).

[22] Emily J Richardson and Mick Watson. "The automatic annotation of bacterial genomes". In: *Briefings in bioinformatics* 14.1 (2013), pp. 1–12.

[23] Torsten Seemann. "Prokka: rapid prokaryotic genome annotation". In: *Bioinformatics* 30.14 (Mar. 2014), pp. 2068–2069. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu153. URL: https://doi.org/10.1093/bioinformatics/btu153.

[24] *Support Vector Machines with examples*. Scikit-learn. 2020. URL: https://scikit-learn.org/stable/modules/svm.html (visited on 05/23/2021).

[25] *What are proteins and what do they do*. U.S. National Library of Medicine. 2021. URL: https://medlineplus.gov/genetics/understanding/howgeneswork/protein/ (visited on 04/13/2021).