Norwegian University
of Life Sciences

Master's Thesis 2024 30 ECTS
Faculty of Science and Technology (REALTEK)

Tsetlin machine for classifying genetic data from sea-floor species

Halvor Hauge Steffenssen
Data science

Preface

I want to thank all the people who have made my years at NMBU the best of my life: my housemates in "Gokk," who make me never want to live alone, and my good friends in Lærken, who have given me a place I can always return to. You have all made me a different and better person than I was when I started here at NMBU.

I also want to thank my supervisor, Kristian Hovde Liland, and co-supervisor, Lars-Gustav Snipen. They both have pushed me to finish this thesis. Their guidance throughout this whole process has been invaluable.

Lastly, I want to thank my parents, who always believe in me and have my back no matter what.

Abstract

With the amount of genetic data we can extract from nature with modern sequencing technology, there is a growing need for tools to help classify and analyze this data. Machine learning algorithms like Random Forest and Artificial Neural Networks are already in use in this field of bioinformatics.

Tsetlin Machine is a new type of machine learning that has shown much promise in DNA classification. It uses binary representation and logic that are close to how a computer operates to create models. This thesis will try to test the Tsetlin Machine's ability to classify genetic data.

A database with the DNA of 709 species commonly found in deep-sea sediments that were picked based on the results of the AQUAeD project. Will be split up into different datasets. The Tsetlin Machine, together with a random forest model, a Convolutional neural network, and a model that counts the number of GC bases, gets these datasets and tries to classify different classes on multiple taxonomic ranks. They are then evaluated based on the accuracy of their classification and the speed of training.

The results show that the Tsetlin Machine has great promise in this field and acquired similar scores to the Random Forest Classifier and the convolutional Neural Network in accuracy and speed.

Abstract

Med den voksende mengden av genetisk data vi kan hente ut fra naturen med moderne sekvenserings teknologi, er det en økt nødvendighet fro verktøy som kan hjelpe med klasifisering og analyse av denne dataen. Maskin lærings algorytmer slik som Random Forest Classifier, og Artificial Neural network, er allerede i bruk i dette bioinformatikk feltet.

Tsetlin Maskin er en ny type maskinlæring som har vist seg å vere lovende i DNA klassifiserings feltet. Det bruker biner representasjon og logikk som er nerme slik en datamaskin opererer, for å lage modeller. Denne oppgaven vil teste Testlin maskinens egenskap til å klassifisere genetisk data.

En database med 709 arter som er vanlig i dyphavsediment som var valgt basert på resultatene av AQUAeD prosjektet. vill bli delt opp i forskjellige datasett. Tsetlin Maskinen sammen med en Random Forest modell , et Convolutional neural nettverk og en modell som teller GC baser, får desse datasettene, og prøver å klassifisere forksjellige taxonomiske kategorier. De vil så bli evaluert basert på nyaktigheten på klassifiseringen, og hurtigheten til treningen.

Resultatene viser at Tsetlin Maskiner har veldig lovende resultater, og oppnår lignende resultater som Random Forest Classifier og Convolutionl Neural Nettverket både på nyaktighet og hurtighet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

DNA is the fundamental building blocks of life, with only four bases: adenine (A), cytosine (C), guanine (G), and thymine (T). It encodes the genetic information for millions of different species. With modern sequencing machines, we can extract enormous amounts of data from DNA. This data is usually in the form of reads and short DNA sequences. These reads are combined into complete genomes using a lot of computational power. Often, if we have a lot of genetic data, multiple species are involved, and it's very useful to classify this genetic data before it's combined into complete genomes. Machine learning is a natural choice to classify this genetic data. This thesis will test out a new type of machine learning called Tsetlin Machines for this task. A Tsetlin Machine uses binary data and logic to classify data in a way that is very similar to how a computer works. The Tsetlin machine has been shown to be very effective in Tabular data [1], image data [2], natural language data[3], and has also shown promise on other DNA-based tasks because of DNA's ability to be described easily in binary [4]. The Tsetlin Machine will be compared to other machine learning methods to see if it is as good or better at classifying genetic data.

1.1 Objectives

This thesis aims to test whether the Tsetlin Machine can reliably classify raw genetic data into different taxonomic categories. It will be compared to other classification methods to see if it can outperform them in terms of time taken and accuracy. So, the questions I will try to answer in the thesis are
1. Is the Tsetlin Machine more accurate than other machine learning methods at classifying DNA reads and contigs?

2. Can the Tsetlin Machine be trained faster than other machine learning methods?

A Random Forest Classifier and a Convolutional Neural Network will be used for the other classification methods. Also, as a baseline, a nonmachine learning method that counts the number of GC base pairs in the DNA and classifies them based on the number, which will be called the GC counter, will also be used. The DNA data will be separated into classes based on taxonomy ranks to see if the results differ based on the different ranks.

Chapter 2

Theory

This chapter will cover the theory of the different classifiers, starting with the Tsetlin Machine (TM). The theory on the TM will be based on this article [5], unless another source is specified. Then, the theory for the Random Forest Classifier (RFC) and the Convolutional Neural Network (CNN). The chapter will also cover what taxonomy is and how it is used in this thesis.

2.1 Tsetlin machine

The Tsetlin machine (TM) is a new type of machine learning that tries to predict the class of the data using only binary inputs and very simple logic. Since DNA is very close to binary in its form and can easily be described in four bits or even two, it's a good fit for a TM. To understand a TM, it's essential to start from the bottom and go up, and the basic building block of a TM is the Tsetlin Automaton.
A Tsetlin automaton (TA) is a fixed finite state automaton, a mathematical model used to analyze models with discrete states and the transitions between those states. An example of a 2N-state TA is in Figure 2.1

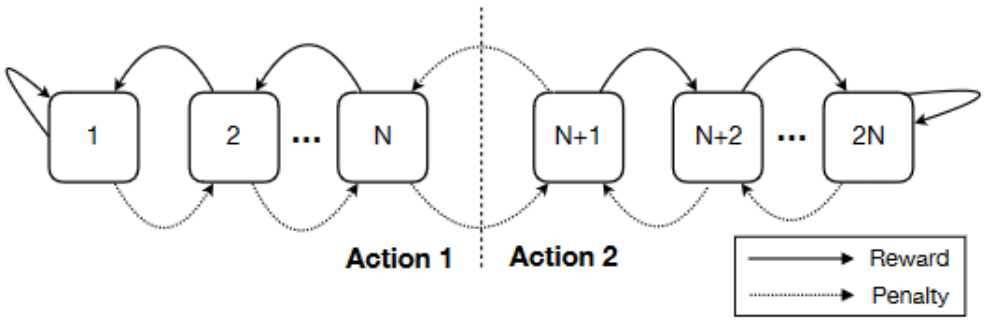

Figure 2.1: A Tsetlin Automaton, figure taken from [5] with permission.

These automatons are put together into clauses, where multiple automatons work together to form an output from an input. So, in the example in Figure 2.2, two clauses have the same inputs. X_1, X_2 and the negated version of X_1 and X_2, the negated version is the opposite of the value, so if X_1 is one, then $\neg X_1$ is 0. Each of these individual inputs is called literals. If the Automatons state is above the threshold, it will be included, if it's below, it will not. If all the included TAs has an output of 1, the clause will give the output 1, if it's not, it will give the output 0. Each clause also has a polarity, denoted by + or -. The + polarity is used to classify

y = 1, and - polarity is used to classify y = 0. The + polarity clauses are summed together in the next layer, and the -polarity clauses are subtracted. If the total sum is ≥ 0, the total output for the TM is y = 1. If the sum is < 0, the output of the TM is y = 0. The different clauses are meant to find different sub-patterns in the input data so it can find the difference in the input data that is typical for the class y = 1 and the class y = 0.

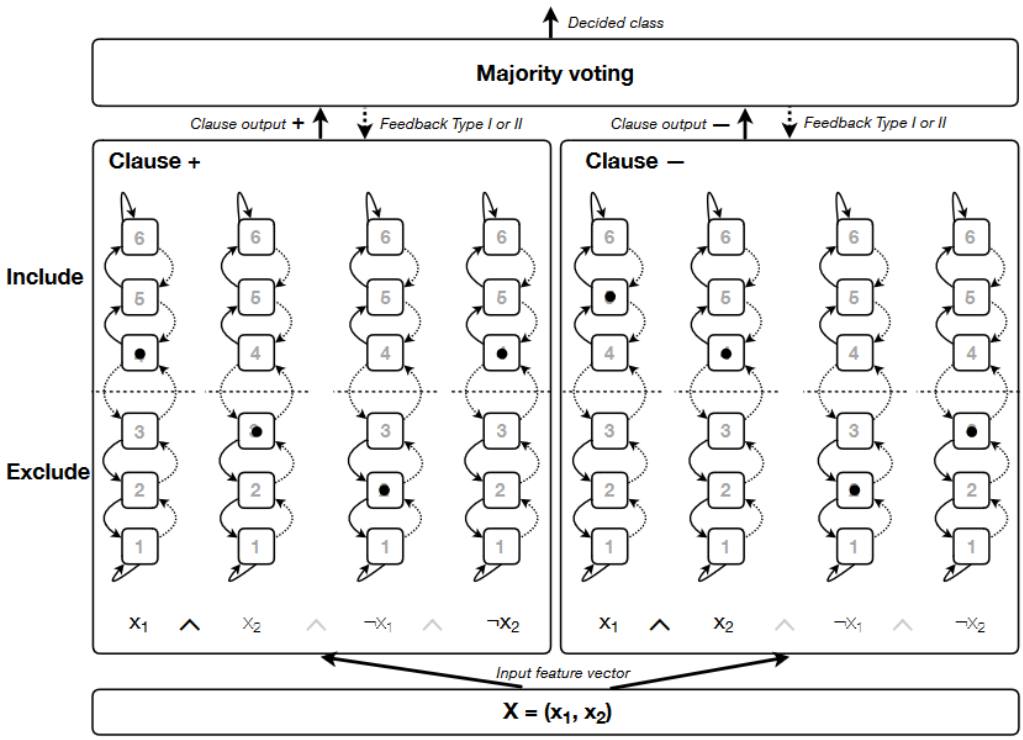

Figure 2.2: Clauses in a Tsetlin Machine, figure taken from [5] with permission.

When you initialize the TM, the TAs in the clauses are randomized. The training process then follows the flowchart 2.3

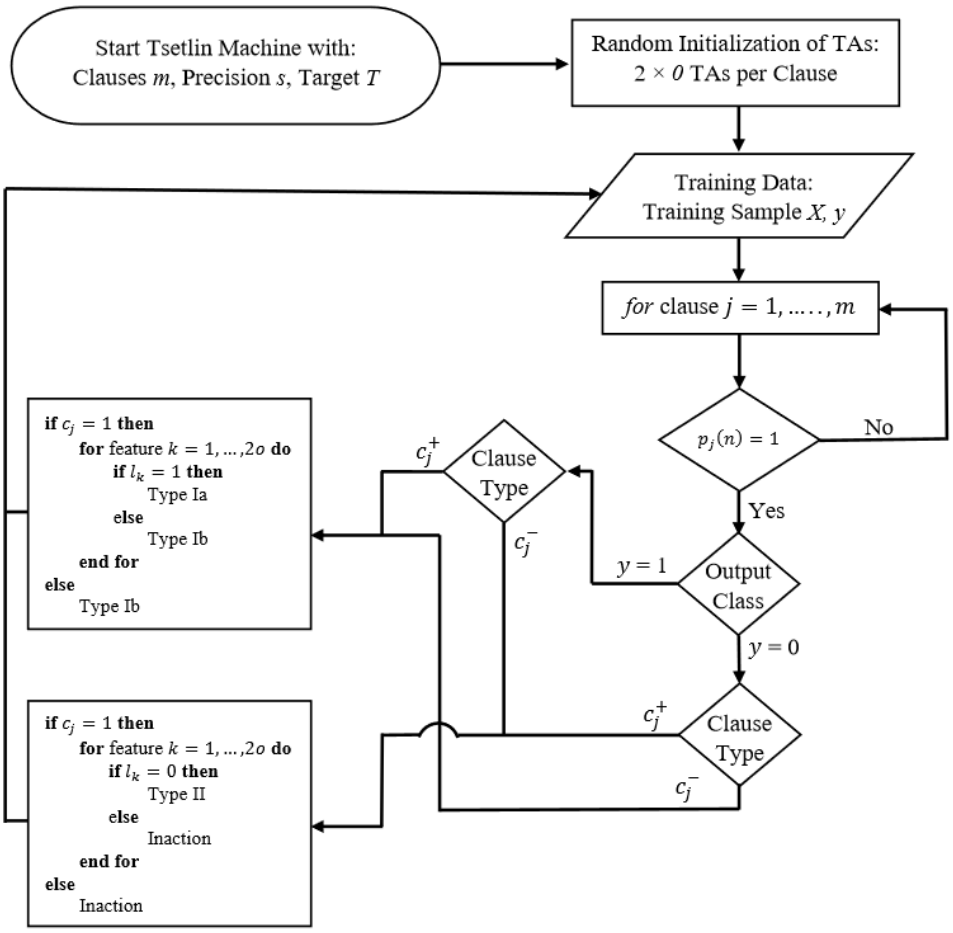

Figure 2.3: Flowchart for the learning process figure taken from [6] with permission.

So when the TM is initialized, it has random clauses that have a chance to update its parameters every time a training sample runs through it. Either Type I feedback or Type II feedback. Type I reinforces true positive outputs, while Type II discourages false positive outputs. To discourage too many clauses from learning the same sub-pattern, a stochastic choosing process limits the number of clauses that learn the same pattern to T. More precisely, it uses this formula to find the number of clauses that get updated if y = 1

$$\frac{T - clip(\sum_{j=1}^{\frac{n}{2}} C_j^1(X) - \sum_{j=1}^{\frac{n}{2}} C_j^0(X), -T, T)}{2T}$$

and this if y = 0

$$\frac{T + clip(\sum_{j=1}^{\frac{n}{2}} C_j^1(X) - \sum_{j=1}^{\frac{n}{2}} C_j^0(X), -T, T)}{2T}$$

So say you have a + polarity clause that has four literals x_1, x_2, $\neg x_1$, and $\neg \neg x_2$, and the input to the clause is (1,0,0,1), and the clause correctly classified y = 1. Then, the clause would get type I feedback. So the individual TAs that had a one as input then get type Ia feedback, the position of the state of the TA will move towards the include part of the automaton with a probability of $\frac{s-1}{s}$. If the individual TA had 0 as an input, then it gets type Ib feedback, and the TA will change towards the exclude part of the TA with a probability of $\frac{1}{s}$. The same feedback will happend for TAs in - polarity clauses that classify y = 0.

If you have a - polarity clause that classifies y = 1, the clause will get type II feedback. So if the input to the literal was 0, the TA will move towards include with a probability of 1. If the input to the literal was 1, nothing happens to the TA.

The hyperparameters T and s can be set as anything, but the formulas used for s and T in this

thesis are.
$T(C) = \sqrt{\frac{C}{2} + 2}$,
and
$s(C) = 2.534 * ln(\frac{c}{3.7579})$.
Where C is the number of clauses.
These formulas are found to be optimal values for these hyperparameters [7].

2.2 Random forest classifier

To understand Random forest, it helps first to understand decision trees in general. A decision tree is a classifying method that splits the data until it can classify all the data correctly into one class in every end node [8]. So, for a simple example, let's say you want to make a decision tree that decides whether you have time to eat breakfast.

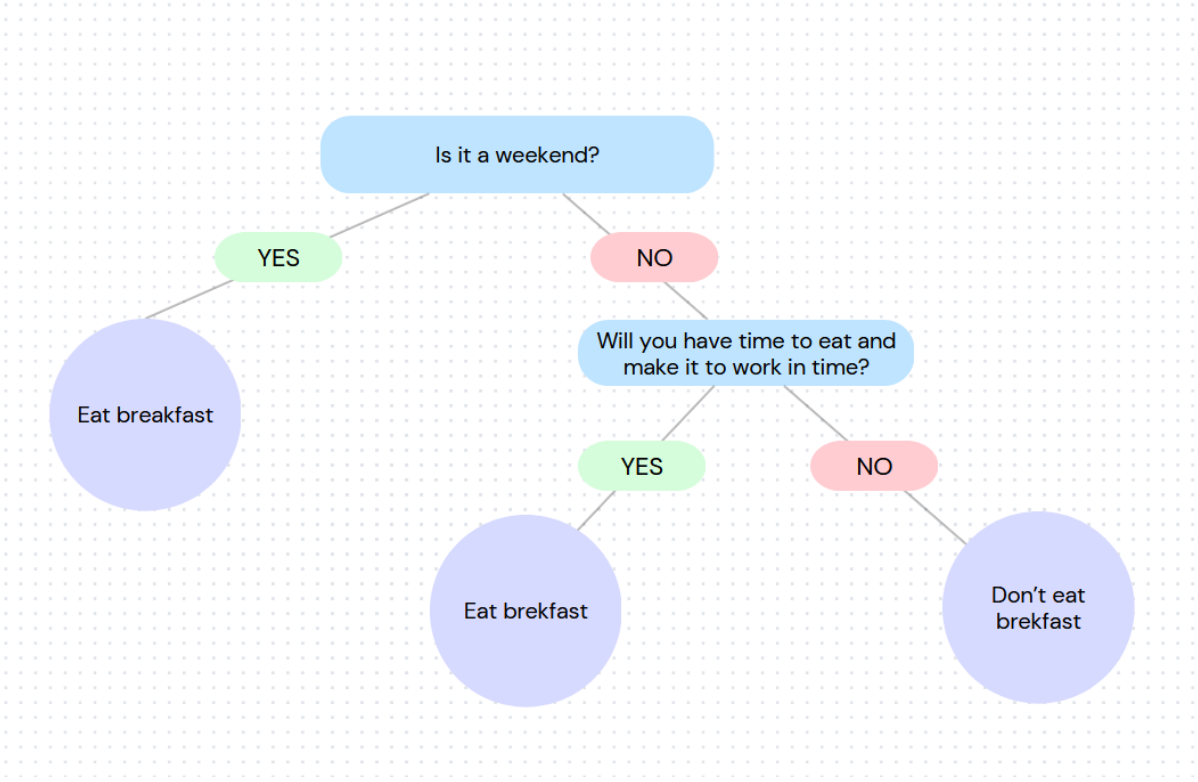

Figure 2.4: Decision tree for if you have time to eat breakfast

If the only data points you had were whether it was a weekend and whether you had time to eat, then this decision tree would correctly classify whether you should eat breakfast every time. This will not work on bigger datasets, so an RFC takes a big dataset, splits it into smaller subsets, and makes decision trees based on the subset of the data. So, for example, consider this made-up data for if a person will go on a run in Table 2.1.

Weather	Mood	Temperature	Time of Day	Going For A Run
Sunny	Happy	Warm	Morning	Yes
Sunny	Happy	Warm	Afternoon	Yes
Rainy	Sad	Cool	Evening	No
Sunny	Sad	Hot	Morning	No
Cloudy	Happy	Mild	Afternoon	Yes
Rainy	Happy	Cool	Evening	No
Cloudy	Sad	Mild	Morning	Yes
Sunny	Happy	Hot	Afternoon	Yes
Rainy	Happy	Cool	Morning	No
Cloudy	Sad	Mild	Evening	No

Table 2.1: Made up data that shows when a person is going for a run

The RFC picks a random subset of the data, so in this example, it takes the first, third, fourth, and sixth rows and only includes the first and second columns and, of course, the classifying column. It then creates a decision tree based on the subset shown in Figure 2.5.

Weather	Mood	Going For A Run
Sunny	Happy	Yes
Rainy	Sad	No
Sunny	Sad	No
Rainy	Happy	No

Table 2.2: Subset of made-up data that shows when a person is going for a run

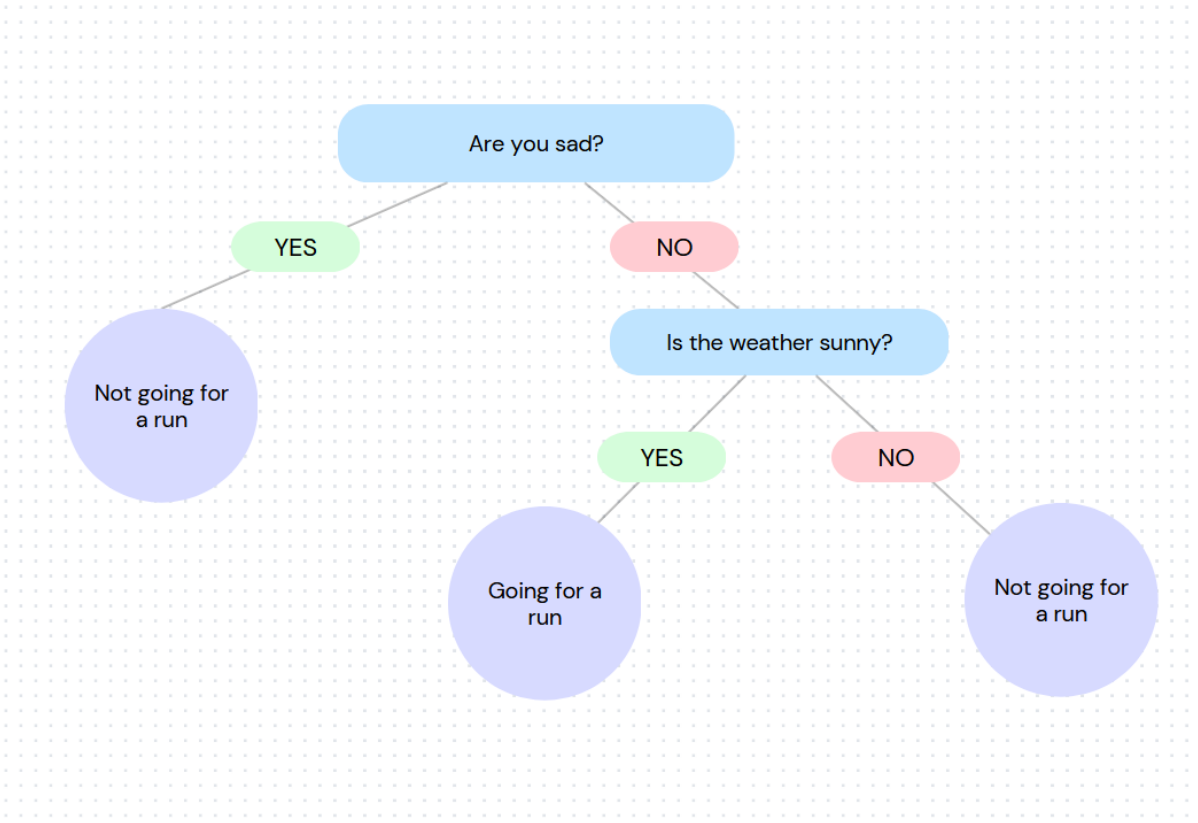

Figure 2.5: Decision tree for the subset

The RFC creates many data subsets, creating many decision trees from the subsets. In general, the more subsets you split the data into, the more accurate the RFC will be, but it will be harder to interpret what features the RFC is making its decision based on. When the model is trained it has all these subsets, with all these decision trees made from them. Then when you send a new data point through the RFC. The RFC will send the data point through all of its decision trees and see what they predict the class will be, and then it takes the majority vote and classifies the new data that way[9].

2.3 Convolutional Neural network

2.3.1 Perceptron

To understand Convolutional Neural Networks (CNN), it's important to start at the bottom and work up, so we start with the basic building block of a neural network, the perception.
The perception takes n inputs x_n, adds a number to the x called a weight w_n, sums up all the numbers, and checks if they meet a threshold. [10]

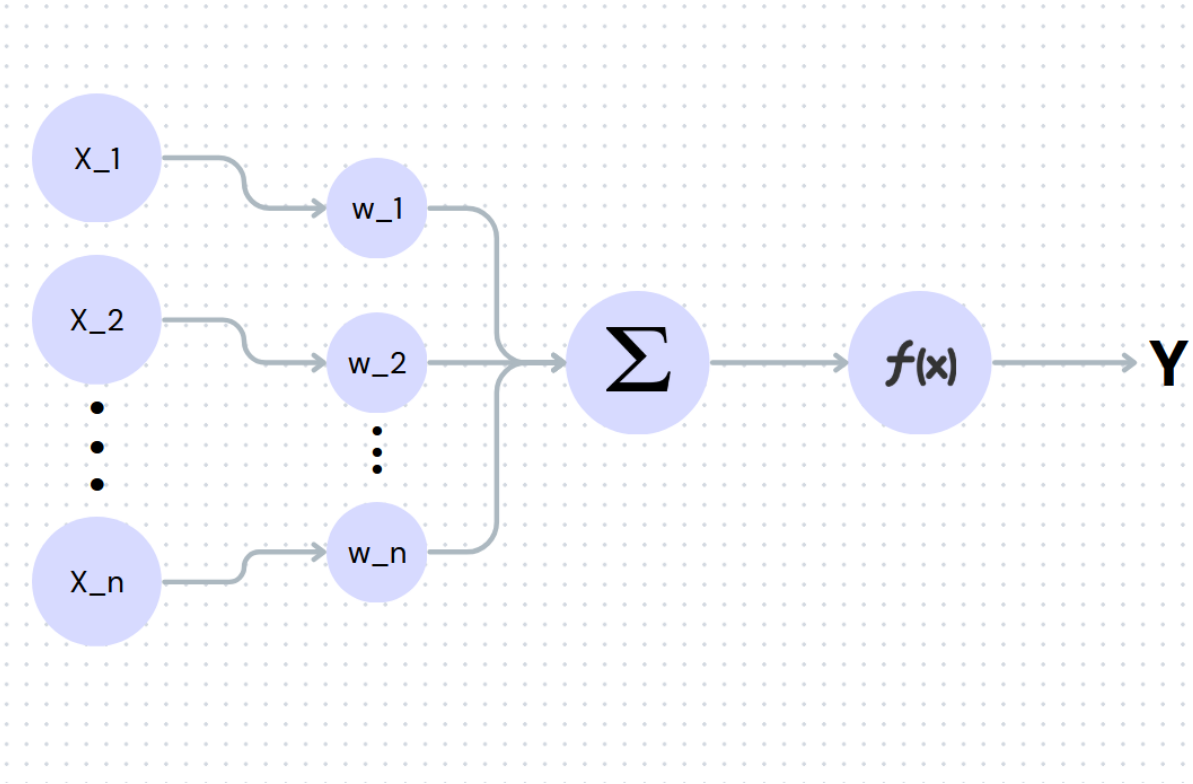

Figure 2.6: A model of a perceptron

it's described by this function $O(x) = f\left(\sum_{i=1}^{m} \omega_i x_i\right)$
Combining multiple perceptions creates a layer of a perception, and combining these again creates a multi-layer perception. So, every input goes to every perception in the layer, and the output goes into another layer, as shown in Figure 2.7. Each of these layers is called a dense layer, the most basic layer in a Neural network. [11]

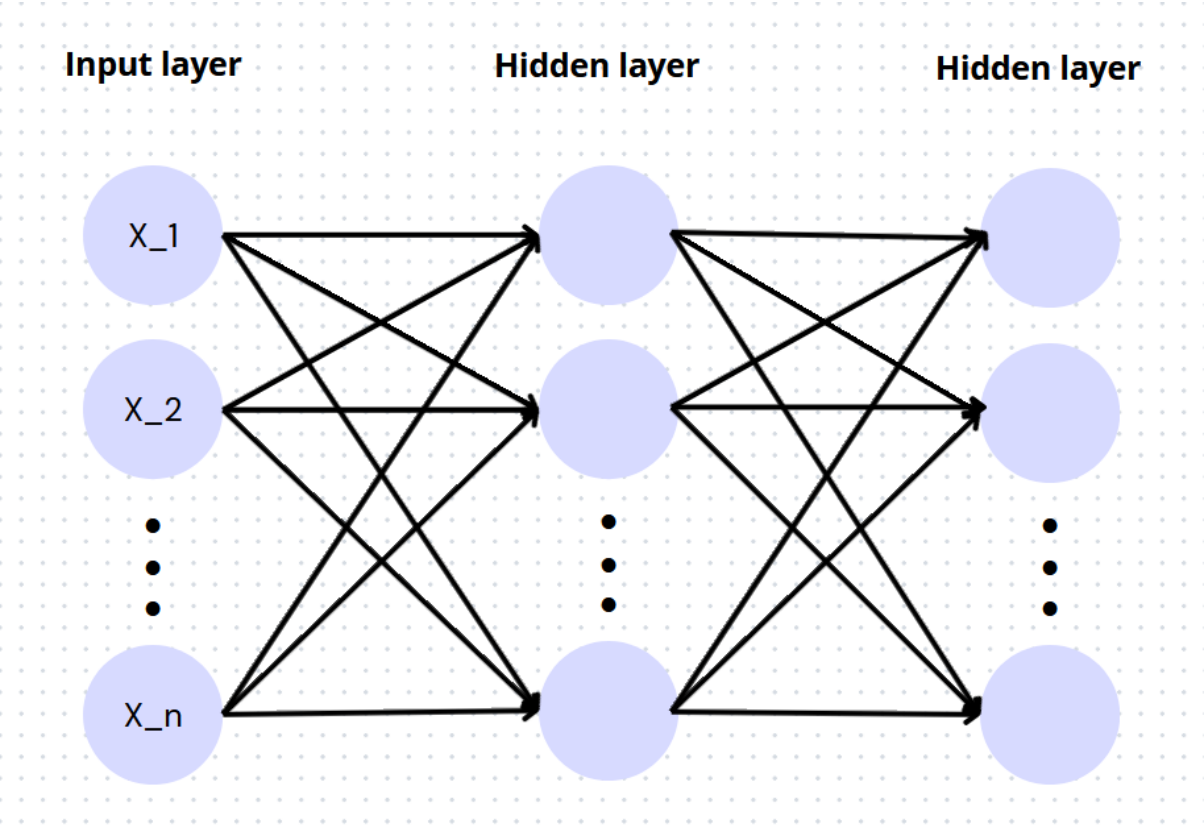

Figure 2.7: A Multi-Layer Perceptron, with two hidden layers.

2.3.2 Activation functions

The activation function of a layer decides the output from the nodes in that layer. Different activation functions are used in different circumstances and network types, as shown in Figure 2.8 and Figure 2.9 [12].

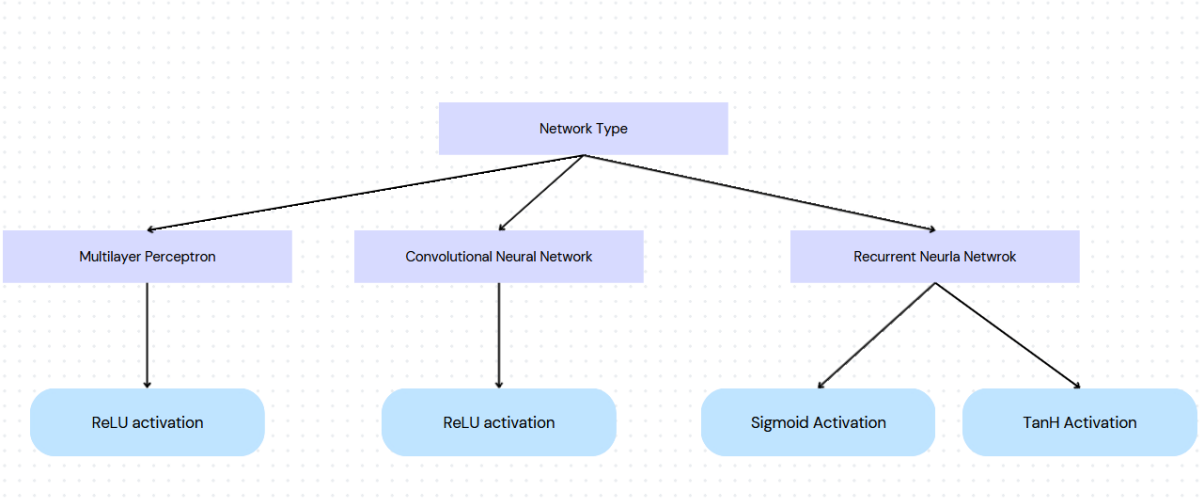

Figure 2.8: Normal Activation functions for hidden layers in different types of neural networks, Figure inspired by [12]

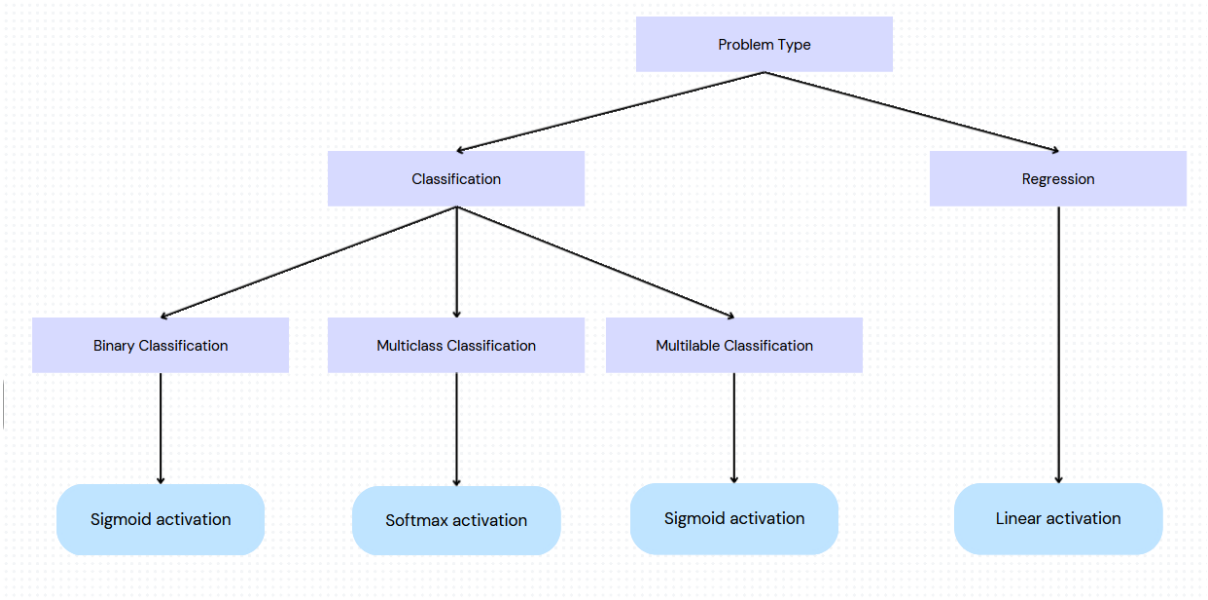

Figure 2.9: Normal Activation functions for output layers in different types of neural networks, Figure inspired by [12]

Since the type of network used in this thesis is a Convoluted Neural Network, The activation function used in the networks hidden layer is the REctified Linear Unit (ReLU). The ReLU function is described by this function.
$ReLU(x) = Max(0, x)$
Since the type of classification used in this thesis is a Binary Activation, the activation function used in the output layer is the Sigmoid function. Which is described by this function
$\sigma(x) = \frac{1}{1+e^{-x}}$

2.3.3 The convolutional part

In a neural network, a convolutional layer consists of filters that slide over the data, multiply the weights of the filter and the data, and sum the multiplications together. The filters can find patterns or features in the data. Since multiple filters slide over the same data in the Convolution layer, the output of this layer becomes higher-dimensional and is called a feature map. [13]

2.3.4 Global max pooling

A global max pooling layer takes high dimensional data and reduces the dimensionality by taking the largest value of the feature data. This makes the data keep the features the convolutional layer sees as the most important, while the rest is not kept. This helps reduce the complexity of the data while keeping the most important information [14]

2.3.5 Validation

There is no validation data in this thesis. This is because the datasets are very large, and the effectiveness of the classifiers can be decided without including validation data.

2.4 Taxonomy

Taxonomy is the science of naming, classifying, and describing all species of living organisms. It tries to classify all living organisms into different ranks based on how closely related they are genetically. A cat, for example, is part of the same class as humans, the class Mammalian, but the orders that cats and humans belong to are different. Together with bears, dogs, and more, cats are part of the Carnivora order. Humans, monkeys, and other apes are part of the Primate's order. The taxonomy used in this thesis is based on the NCBI Taxonomy and uses for prokaryotes seven different classification levels. The different levels of taxonomy used are shown in Figure 2.10, where all the different levels for the species Serratia Marcescens are shown.

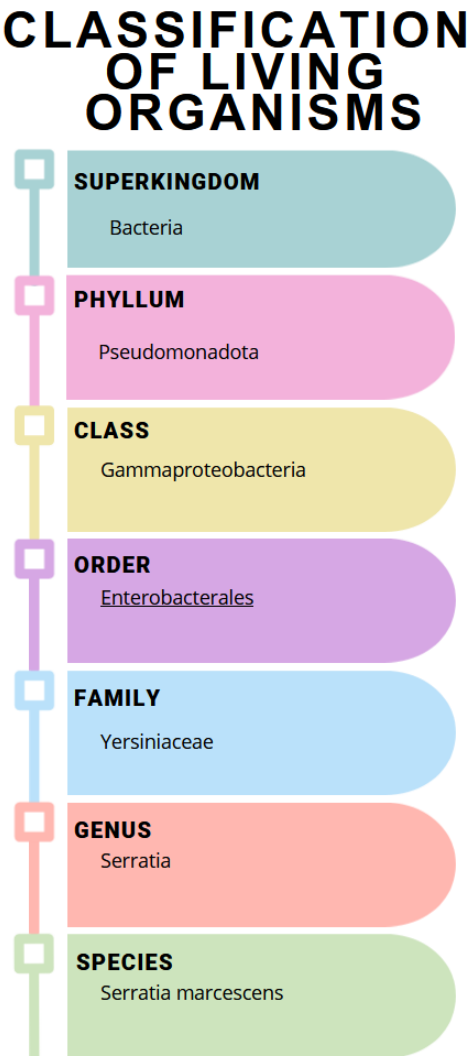

Figure 2.10: The levels of taxonomy used in this thesis. All the levels of the species Serratia Marcescens

The type of data used in this thesis is pure DNA data in the form of text, which will be called DNA strings. It is generated from the complete DNA of species that are already sequenced. Strings from 20 letters long, also called 20 base pairs (bp) long, all the way up to 10,000 letters long, also called ten kilobase pairs long (kbp). If this were real-world data, it would be based on reads, which are short DNA strings extracted by many different methods from 100-500 bp usually, and those would be put together to create contigs, which can be many kbp long [15]. In this thesis, the different inputs to the models will be referred to as DNA stings since they are not reads and contigs from real-world data.

Chapter 3

Method

This chapter will cover how the data was collected and processed, how each of the classifying models will be created, and what programs are used to assist in writing this thesis.

3.1 The data

3.2 Data encoding

The encoding of the data input is very important for machine learning methods to work best. In this thesis, the data for the machine learning models will be encoded in two ways: direct and K-mer encoding.

3.2.1 Direct encoding

In this context, direct encoding means encoding text data to binary data. So, each letter is encoded into 4 bits. A is represented by 0001, C is 0010, G is 0100, and T is 1000. This method will be used by one of the Tsetlin Machines and the Convolutional Neural Network.

3.2.2 K-mer encoding

K-mer encoded data takes the input data and creates each possible K-letter sequence of that data. So if K is five like in the models in this thesis and DNA can be four different bases, then it creates 1024 $4^5 = 1024$ different 5-mers and counts the number of times each 5-mer is in the data. This loses the positional information, but it can still keep some of it since if the 5-mer "AACGA" and "ACGAC" both are in the data, there is a four-letter overlap, so extracting positional features from that is possible. The K-mer method will be used by one of the Tsetlin Machines, and the Random Forest Classifier.

3.3 Where the data comes from

The data is from a collection of DNA sequences picked from species common in sea sediments. The species was picked based on the results of the AQUAeD project, which found species we can say with assurance are pretty common in sea sediments. These species' genetic data and taxonomy were then found with the help of the RefSeq database and the NCBI Taxonomy database. The genetic data of 709 common species found in the sea bed will be used to create DNA strings of different lengths. The models can be trained to classify from all different taxonomic ranks. The data is in Fasta files.

3.4 Pre-processing

The 709 Fasta files will be divided into six different pairs of classes: into the empires: eukaryotes and prokaryotes. Into two different domains: archaea and bacteria. Differentiating the phylum Actinomycetota from the other phyla, differentiating the species Escherichia coli from other bacteria, differentiating the species Serratia Marcescens from the other bacteria, and differentiating the species Methanosarcina Lacustris from the other archaea.
500 DNA strings with the lengths 20, 100, 500, 1000, and 2000 base pairs will be made for each class and sent into the different classifier pipelines.

3.4.1 Tsetlin Machine

There will be two Tsetlin Machine (TM) pipelines, one that encodes the data directly and one that generates K-mers. In the TM pipeline that encodes the data directly, the data will be sent through a function, which creates a binary version of the data where A is represented by 0001, C is 0010, G is 0100, and T is 1000. In the other pipeline, the data will be sent through a sci-kit vectorizer, which vectorizes the input data into characters and creates a vocabulary with all the different k-mers in the input string. It then will check if every k-mer is in the input strings and has a one if it is there and a 0 if not. This will not count the number of times it's there, but only if it's there. Now, in binary, the list will be a data type the TM understands. The data from both pipelines will then be split into test and training data with an 80/20 split. The training data will then be sent through the TM to train it. When trained, it will get the test data and classify it. The results from the TM will then be checked against the real data, and create an accuracy score.
The TM will be trained with different amounts of clauses and with the hyperparameters:
$T(C) = \sqrt{\frac{C}{2} + 2}$,
and
$s(C) = 2.534ln(\frac{c}{3.7579})$.
Where C is the number of clauses [7], different numbers of clauses will be tested to find a good balance between the accuracy score and the time it takes for the test data to run. The accuracy will by far be the most important factor.

3.4.2 Random Forest Classifier

In the Random Forest Classifier, the data will be sent through a sci-kit vectorizer like one of the TM pipelines. Unlike the TM, it counts the number of times each of these K-mers shows up in each input and makes a sparse matrix of the features in each input. It then splits up the data into training and testing with an 80/20 split, trains the RFC with the train data, predicts with the testing data, checks the predicted value compared to the actual value, and creates an accuracy score.

3.4.3 GC content

In the GC content classifier, the code counts the number of G and C input lists, and if it's over 50%, it classifies it into one class, and if it's under 50 %, it's in the other. It then checks the guessed class compared to the actual class and creates an accuracy score.

3.4.4 Convolutional Neural Network

The Convolutional Neural Network will have a one-dimensional convolutional layer that gives the model 128 filters, a kernel size of 7, and the ReLU (Rectified Linear Unit) activation function. This will create a high-dimensional feature map. Then, a global max pooling layer will reduce

dimensionality to help the network find the most essential features. Then, a dense layer with a dimension size of 64 and another ReLU activation function. Lastly, a dense layer with dimension one and a sigmoid activation function will classify the input. The hyperparameters of this model are found mainly through trial and error. Then, the model is trained and tested, and it will return its guesses for the test data, which will be checked against the actual classes and will give the model's accuracy.

3.5 Tools used

The AI "Chat-GPT" has been used to help with coding and debugging the code and to create multiple tables in this thesis. It has been used in accordance with the "Guidelines for the use of artificial intelligence at REALTEK." The program Grammarly, which helps with spelling and grammar has also been used.

Chapter 4

Results

This chapter will go through the accuracy and speed of all the models and compare the different models and datasets. The datasets are mostly the same, but what is being classified differs. The datasets will be named based on what they are classifying. The chapter will begin by gooing through the accuracy the models had on all the datasets, then the time it took to train the models on the different datasets. Lastly, it will go through each dataset and compare the different model's accuracy.

4.1 The datasets accuracy

These graphs show the different classifiers running five times on each input length and the classifier's average accuracy. Figure 4.1 tries to predict whether the input is of the class prokaryote or eukaryote.

4.1.1 Eukaryote or prokaryote

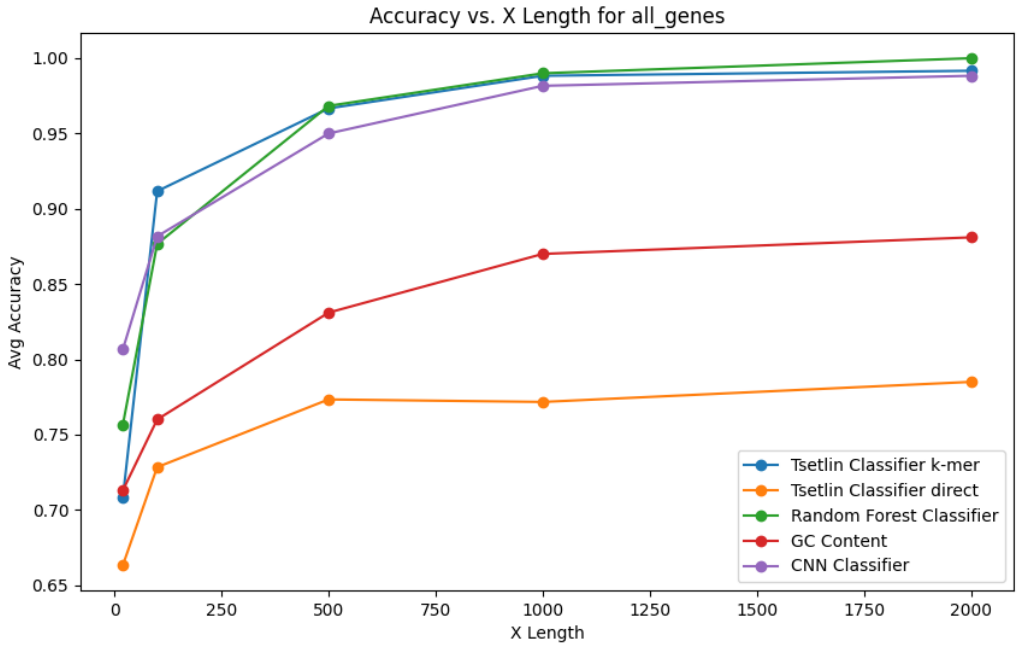

Figure 4.1: The accuracy of 5 different classifiers, with different input lengths, that try to predict if the given sequence is a prokaryote or a eukaryote.

Figure 4.1 shows the models need at least 100-500 bp before the accuracy is acceptable. The TM with K-mer encoding, the CNN, and the RFC are doing the best here. This is true for almost all the datasets.

Figure 4.2 Uses only the prokaryote data and tries to predict if the input is of the class archaea or bacteria.

4.1.2 Archaea or bacteria

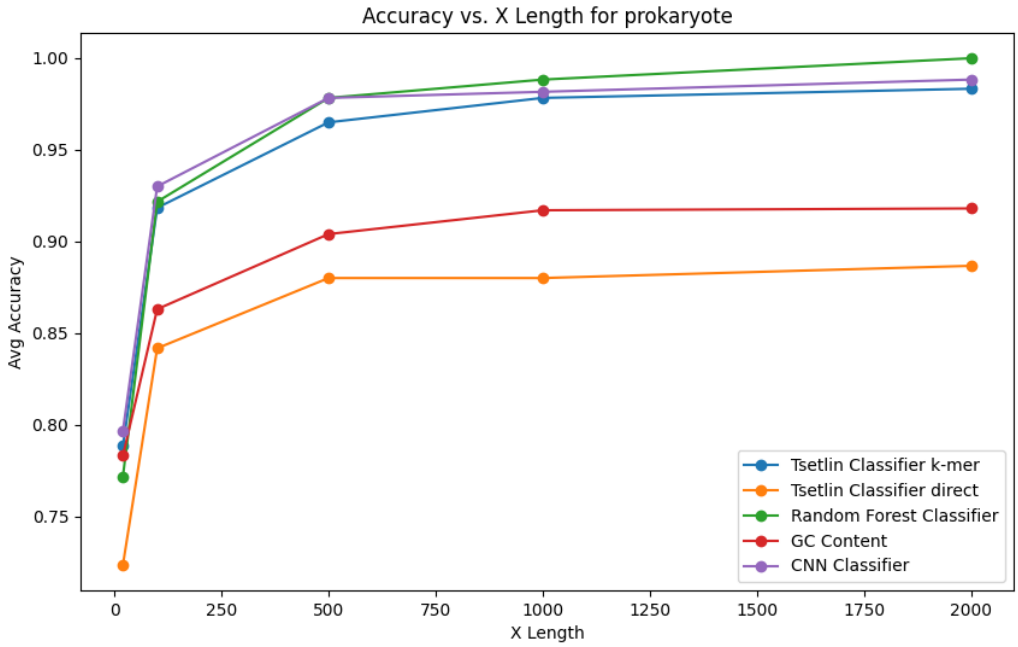

Figure 4.2: The accuracy of 5 different classifiers, with different input lengths, that try to predict if the given sequence is an archaea or a bacteria.

In Figure 4.2, the CG counter is doing very well, it's clear that this is the biggest difference in CG content between the classes.

Figure 4.3 Uses only the prokaryote data and tries to predict if the input is of the phylum Actinomycetota or any other phyla.

4.1.3 Actinomycetota or other phyla

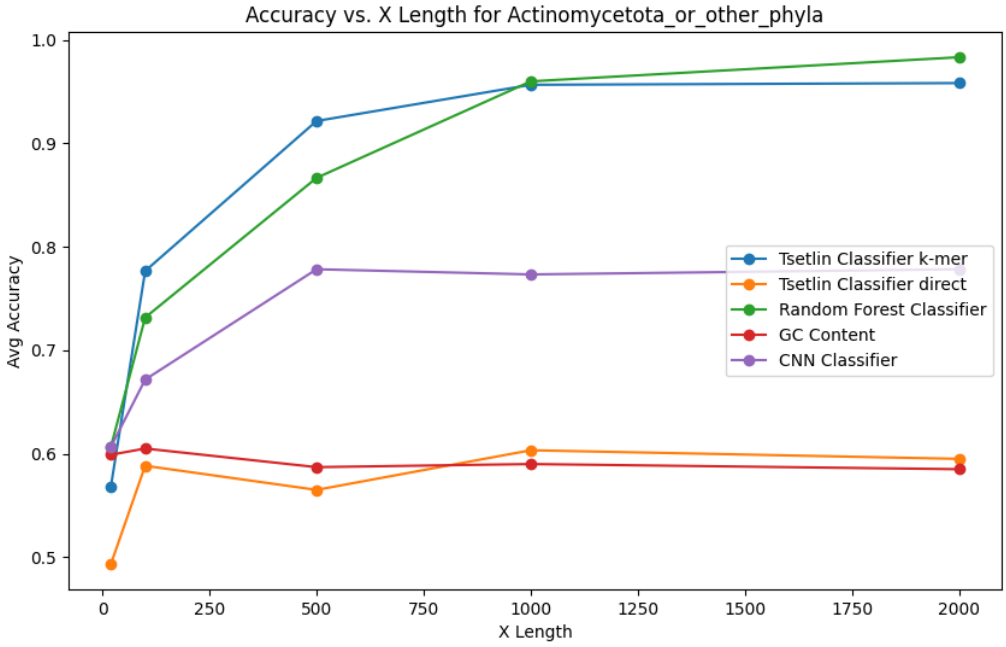

Figure 4.3: The accuracy of 5 different classifiers, with different input lengths, that try to predict if the given sequence is an Actinomycetota or any other phyla.

for Figure 4.3 The CNN is worse than the earlier datasets. Stagnating a little after an input length of 100 bp. The TM with K-mer encoding and the RFC look more similar to the other datasets.

Figure 4.4 Uses only the bacteria data and tries to predict if the input is of the class Escherichia Coli or any otherspecies of bacteria.

4.1.4 Escherichia Coli or other bacteria

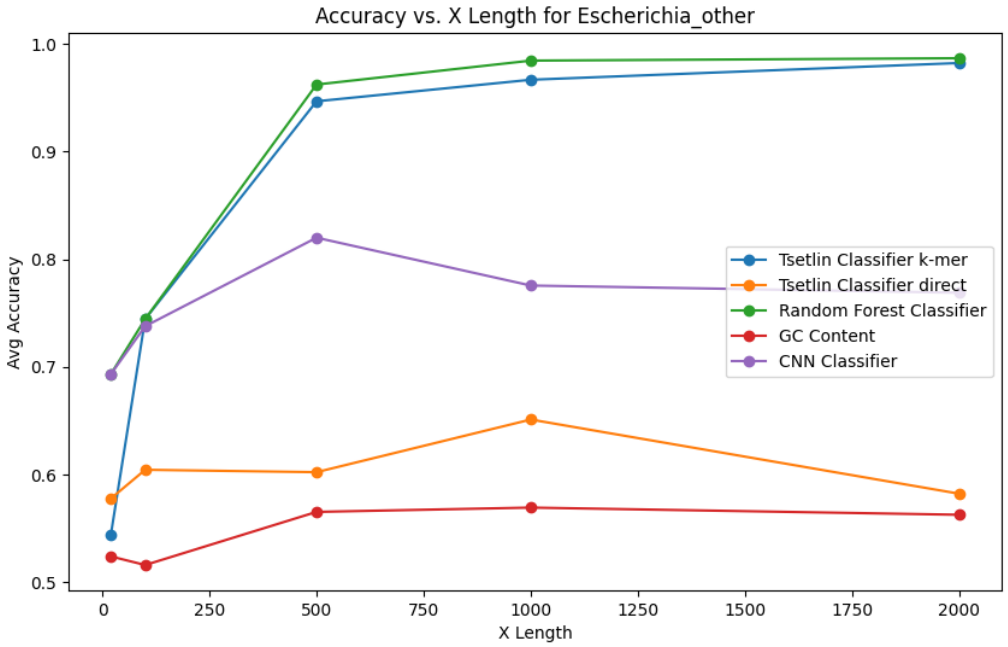

Figure 4.4: The accuracy of 5 different classifiers, with different input lengths, that try to predict if the given sequence is of the class Escherichia Coli or any other species of bacteria.

In Figure 4.4, the CNN is a bit worse than the TM with K-mer encoding and the RFC. Preforming worse on 1000 bp than 100 and 500 bp.

Figure 4.5 Uses only the bacteria data and tries to predict the classes Serratia Marcescens or any other species of bacteria.

4.1.5 Serratia Marcescens or other bacteria

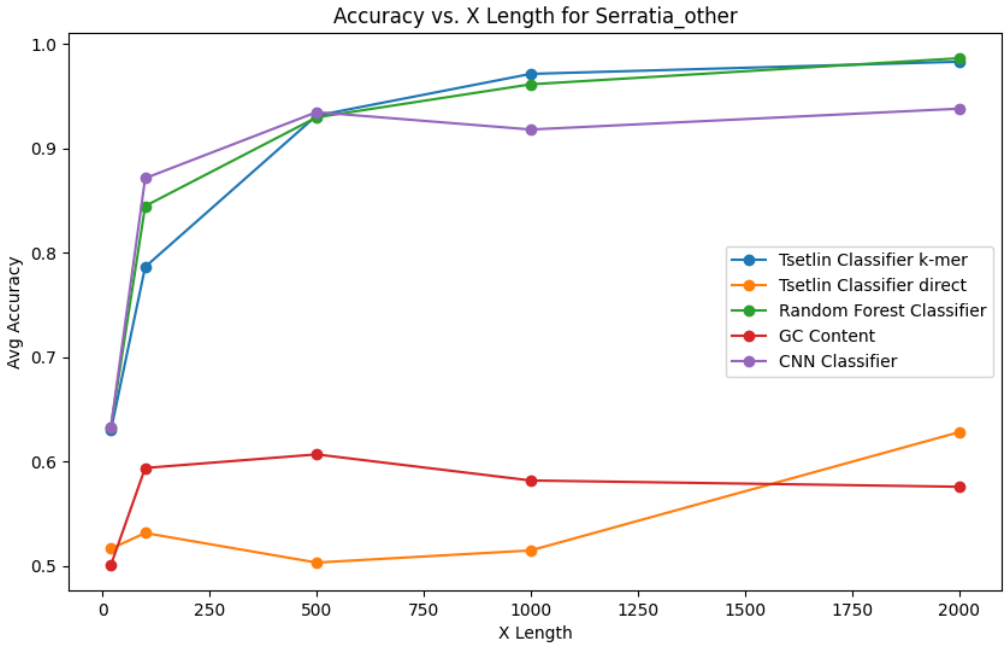

Figure 4.5: The accuracy of 5 different classifiers, with different input lengths, that try to predict whether the given sequence is the Serratia Marcescens or any other species of bacteria.

In Figure 4.5, the CNN is almost the same level as the TM with K-mer encoding and the RFC.

Figure 4.6 Uses only the archaea data and tries to predict the classes Methanosarcina lacustris or any other species of arcaea.

4.1.6 Methanosarcina Lacustris or other archaea

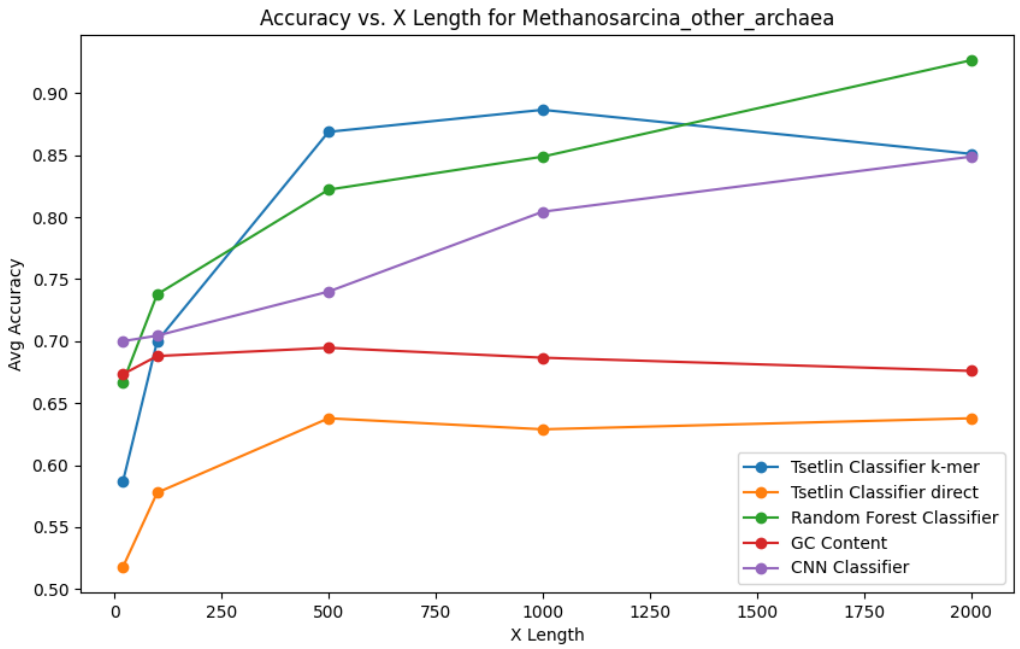

Figure 4.6: The accuracy of 5 different classifiers, with different input lengths, that try to predict whether the given sequence is the Maethaosarcina or any other species of archaea.

In Figure 4.6, the spread of all the classifiers is largest, and also the best classifier, the RFC, is the only one reaching an accuracy of over 0.9 on 2000 bp.

Table 4.1 shows The time it took for five different classifiers, with different input lengths, to try to predict if the given sequence is a prokaryote or a eukaryote. This is the data that is used in Figure 4.1.

X Length	Classifier	Average Accuracy
20	Tsetlin Classifier k-mer	0.708333
	Tsetlin Classifier direct	0.663333
	Random Forest Classifier	0.756667
	GC Content	0.713000
	CNN Classifier	0.806667
100	Tsetlin Classifier k-mer	0.911667
	Tsetlin Classifier direct	0.728333
	Random Forest Classifier	0.876667
	GC Content	0.760000
	CNN Classifier	0.881667
500	Tsetlin Classifier k-mer	0.966667
	Tsetlin Classifier direct	0.773333
	Random Forest Classifier	0.968333
	GC Content	0.831000
	CNN Classifier	0.950000
1000	Tsetlin Classifier k-mer	0.988333
	Tsetlin Classifier direct	0.771667
	Random Forest Classifier	0.990000
	GC Content	0.870000
	CNN Classifier	0.981667
2000	Tsetlin Classifier k-mer	0.991667
	Tsetlin Classifier direct	0.785000
	Random Forest Classifier	1.000000
	GC Content	0.881000
	CNN Classifier	0.988333

Table 4.1: Accuracy of different classifiers for various X Length values.

4.2 The datasets runtime

These graphs show the different classifiers running five times on each input length and show the classifier's average runtime.
Figure 4.7 uses all the data and tries to predict the classes prokaryote or eukaryote.

4.2.1 Eukaryote or prokaryote

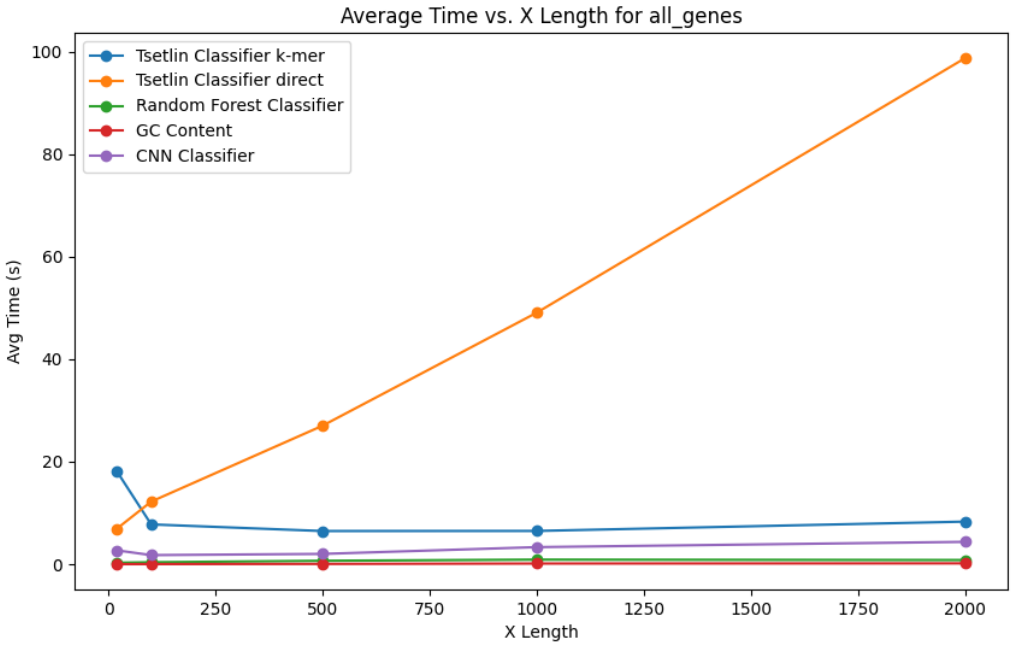

Figure 4.7: The time it took for five different classifiers, with different input lengths, to try to predict if the given sequence is a prokaryote or a eukaryote.

In Figure 4.7, the TM with direct encoding goes up faster than the other classifiers. It will not be included in the other graphs.

Figure 4.8 Uses only the prokaryote data and tries to predict if the input is of the class archaea or bacteria. It excludes the "Tsetlin Classifier direct" since that makes it harder to see the other data.

4.2.2 Archaea or bacteria

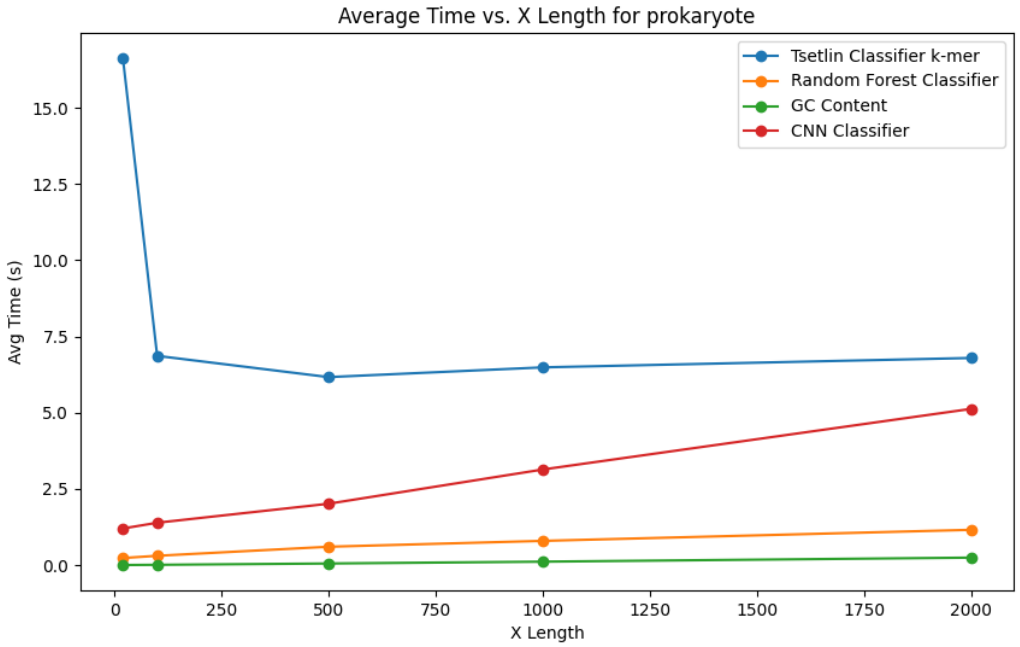

Figure 4.8: The time it took for four different classifiers, with different input lengths, to predict whether the given sequence is an archaea or a bacteria.

Figure 4.9 Uses only the prokaryote data and tries to predict if the input is of the class Actinomycetota or another phyla. It excludes the "Tsetlin Classifier direct" since that makes it harder to see the other data.

4.2.3 Actinomycetota or other phyla

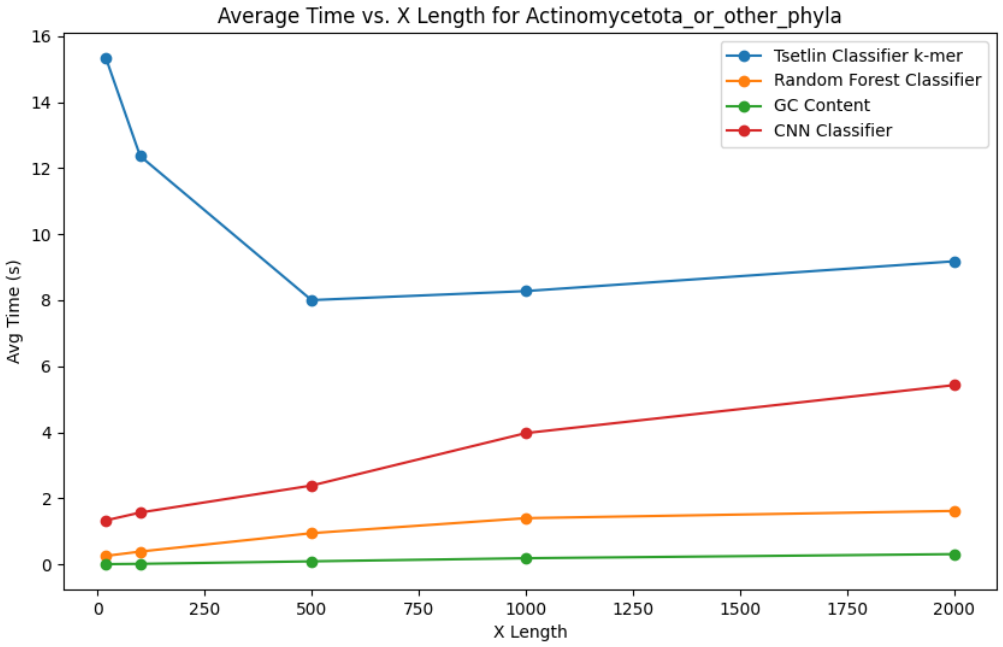

Figure 4.9: The time it took for four different classifiers, with different input lengths, to try to predict if the given sequence is an Actinomycetota or another phylum.

Figure 4.10 Uses only the bacteria data and tries to predict if the input is of the class Escherichia Coli or any other species of bacteria. It excludes the "Tsetlin Classifier direct" since that makes it harder to see the other data.

4.2.4 Escherichia Coli or other bacteria

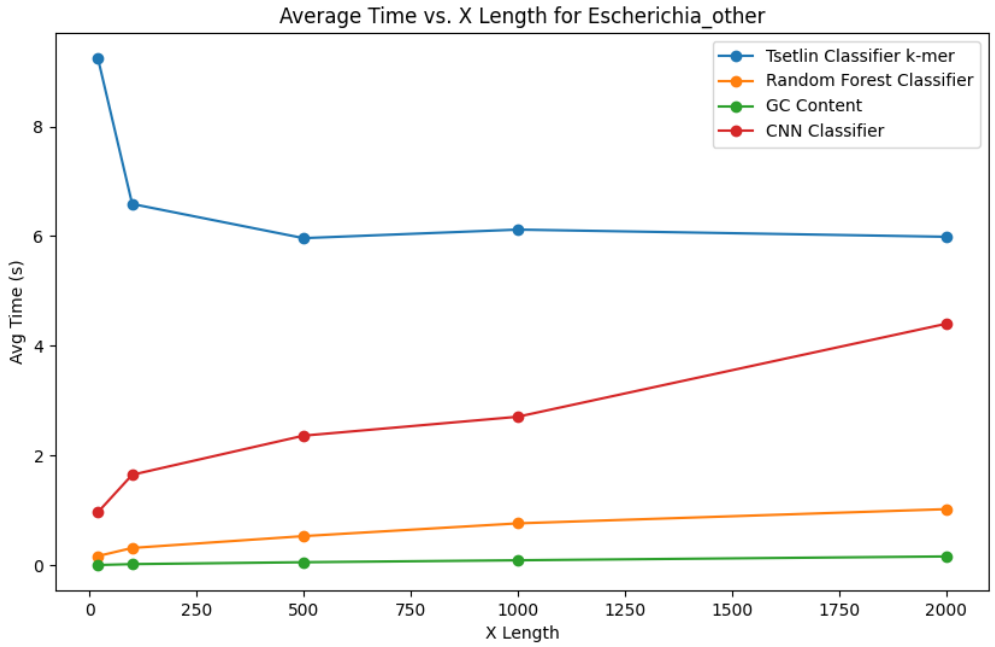

Figure 4.10: The time it took for four different classifiers, with different input lengths, to try to predict if the given sequence is the species Escherichia Coli or if it's any other speciesof bacteria.

Figure 4.11 Uses only the bacteria data and tries to predict the classes Serratia Marcescens or any other bacteria. It excludes the "Tsetlin Classifier direct" since that makes it harder to see the other data.

4.2.5 Serratia Marcescen or other bacteria

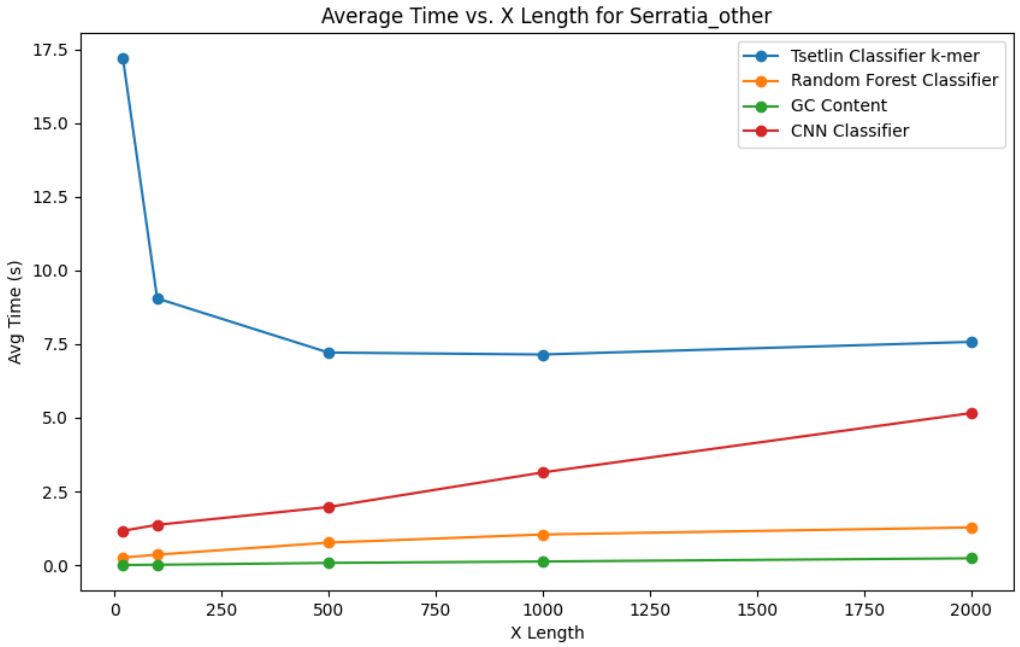

Figure 4.11: The time it took for four different classifiers, with different input lengths, to predict if the given sequence is the species Serratia Marcescens or any other species of bacteria.

Figure 4.12 Uses only the archaea data and tries to predict the classes Methanosarcina Lacustris or other species of archaea. It excludes the "Tsetlin Classifier direct" since that makes it harder to see the other data.

4.2.6 Methanosarcina Lacustris or other archaea

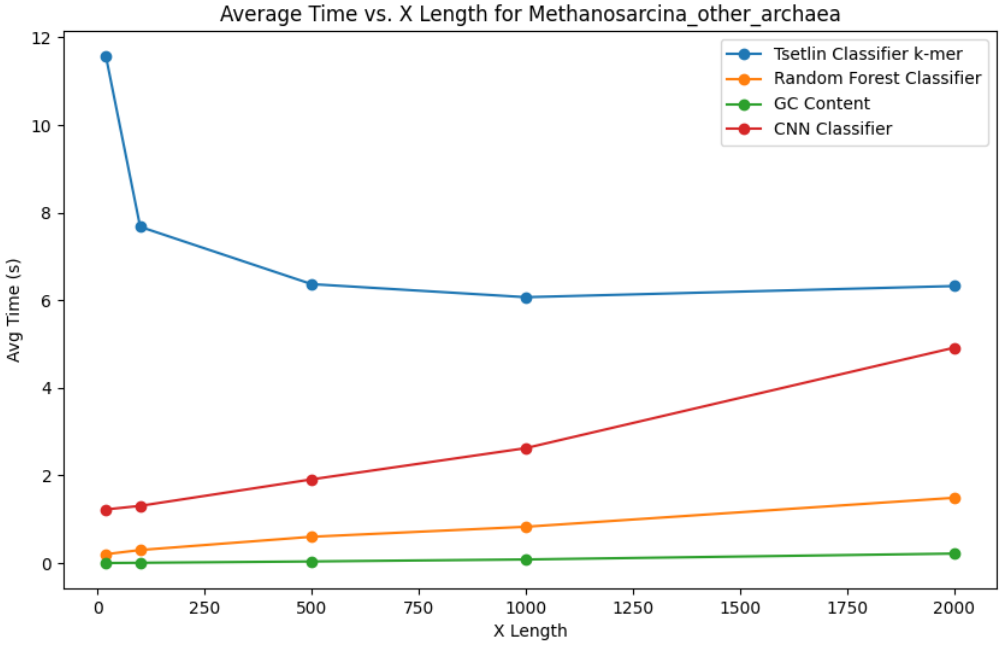

Figure 4.12: The time it took for four different classifiers, with different input lengths, to predict if the given sequence is the species Methanosarcina Lacustris or other archaea.

In all the figures in this section, the time the TM with K-mer encoding uses starts high and drops down. The Tm with K-mer encoding also seems to increase slower than the CNN, which becomes more apparent in Table 4.2, where the classifiers, except the Tsetlin Classifier with direct encoding, are trained with very long input lengths. To see if the trends are different higher up. Even though the accuracy doesn't change much. The table shows the classifiers trying to classify if the input is eukaryotes or prokaryotes with high input lengths.

X Length	Classifier	Average Time (s)
2000	Tsetlin Classifier k-mer	8.726889
	Random Forest Classifier	1.213532
	CNN Classifier	9.199748
4000	Tsetlin Classifier k-mer	9.469364
	Random Forest Classifier	1.816308
	CNN Classifier	10.960591
8000	Tsetlin Classifier k-mer	9.652495
	Random Forest Classifier	2.484141
	CNN Classifier	18.081925
16000	Tsetlin Classifier k-mer	14.728810
	Random Forest Classifier	3.142332
	CNN Classifier	29.668247

Table 4.2: Average time taken by different classifiers for various X Length values.

Table 4.3 shows The time it took for five different classifiers, with different input lengths, to try to predict if the given sequence is a prokaryote or a eukaryote. This is the data that is used in Figure 4.7.

X Length	Classifier	Average Time (s)
20	Tsetlin Classifier k-mer	18.114743
	Tsetlin Classifier direct	6.932466
	Random Forest Classifier	0.285667
	GC Content	0.002370
	CNN Classifier	2.697376
100	Tsetlin Classifier k-mer	7.764133
	Tsetlin Classifier direct	12.210904
	Random Forest Classifier	0.377048
	GC Content	0.012759
	CNN Classifier	1.767590
500	Tsetlin Classifier k-mer	6.469070
	Tsetlin Classifier direct	27.027963
	Random Forest Classifier	0.664114
	GC Content	0.046691
	CNN Classifier	2.004546
1000	Tsetlin Classifier k-mer	6.486216
	Tsetlin Classifier direct	49.093130
	Random Forest Classifier	0.887088
	GC Content	0.123626
	CNN Classifier	3.327825
2000	Tsetlin Classifier k-mer	8.307784
	Tsetlin Classifier direct	98.785298
	Random Forest Classifier	0.809729
	GC Content	0.157267
	CNN Classifier	4.356232

Table 4.3: Average time taken by different classifiers for various X Length values.

4.3 The classifiers on the different data

The graphs in this section show how one classifier performed on all the different datasets. All the classifiers in this section have a 1000-input length and are run five times.

4.3.1 Tsetlin Machine with k-mer encoding

Figure 4.13 shows the Testlin machine with k-mer encoding on all the datasets, with the average, max, and min accuracy.

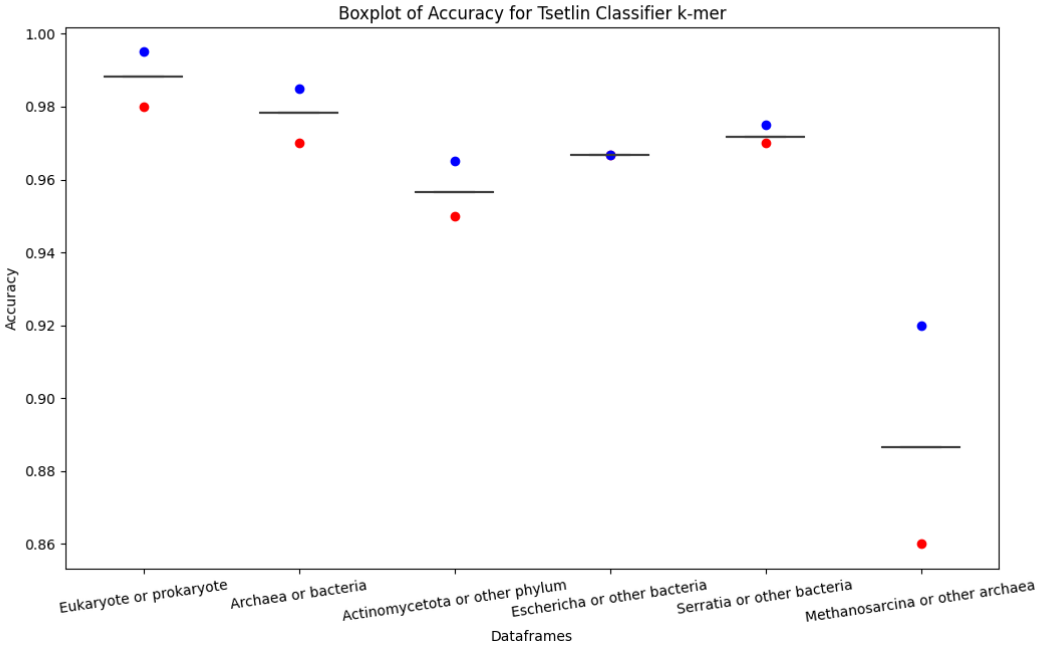

Figure 4.13: The average, max, and minimal accuracy of the Tsetlin Classifier with K-mer encoding, with input length of 1000, on the six different datasets

In Figure 4.13, The Tm is not quite as good at classifying the phyla from each other and even worse at classifying the species Methanosarcina Lacustris from the other archaea.

4.3.2 Tsetlin Machine with direct encoding

Figure 4.14 shows the Testlin machine with direct encoding on all the datasets, with the average, max, and min accuracy.

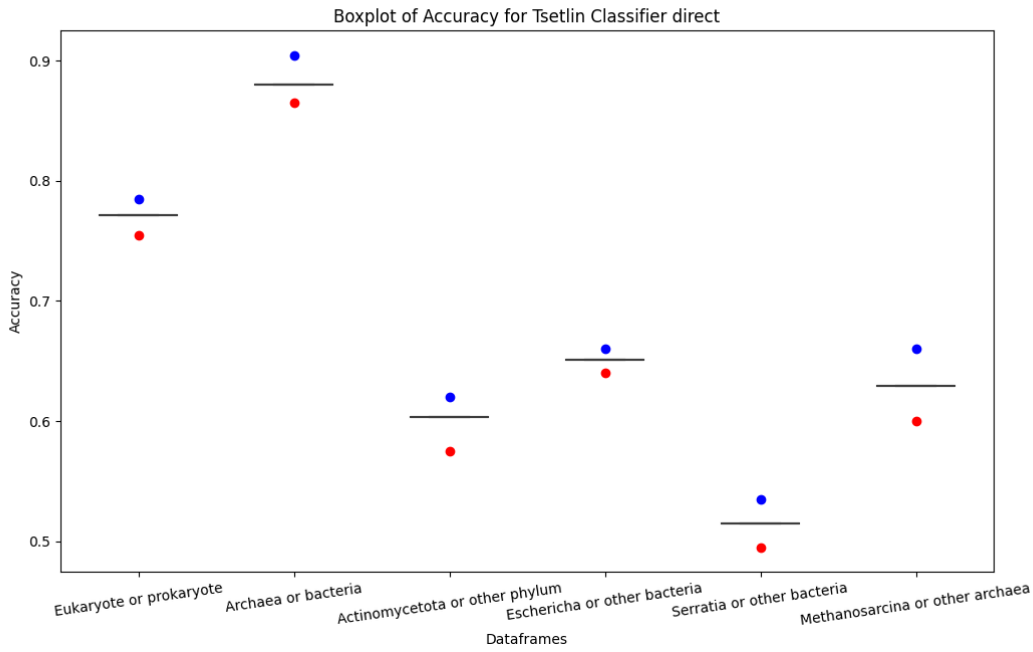

Figure 4.14: The average, max, and minimal accuracy of the Tsetlin Classifier with direct encoding, with input length of 1000, on the six different datasets

In Figure 4.14, the TM is okay at classifying the archaea or bacteria. It isn't good at classifying the other datasets.

4.3.3 Random Forest Classifier

Figure 4.15 shows the Random Forest Classifier on all the datasets with the average, max, and min accuracy.

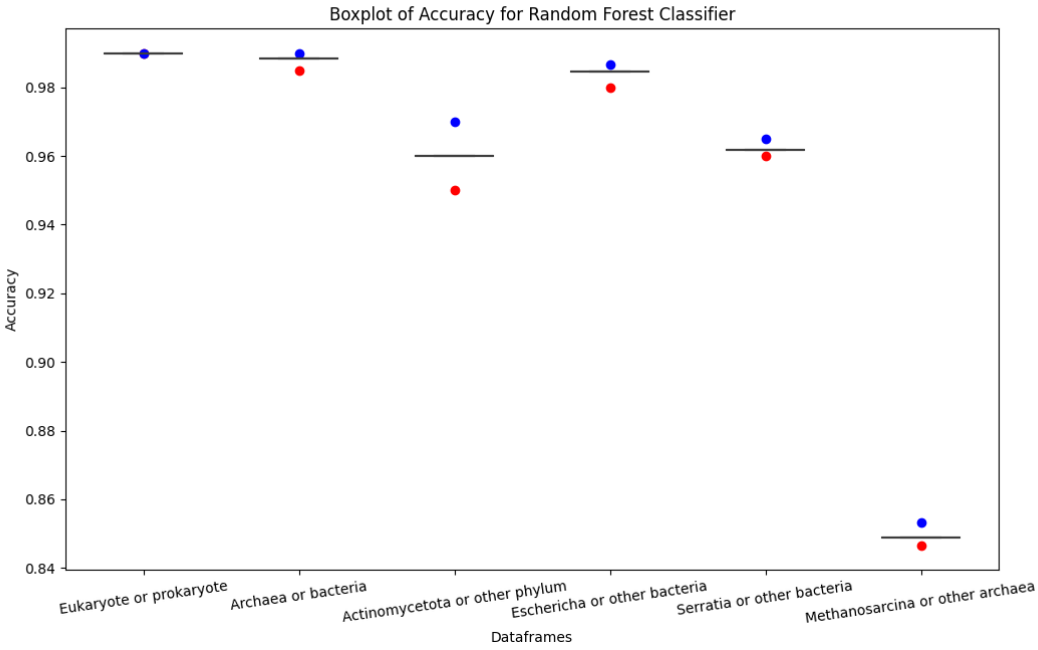

Figure 4.15: The average, max, and minimal accuracy of the Random Forest Classifier, with input length of 1000, on the six different datasets

Figure 4.15 shows that the RFC is consistent in all the datasets, only struggling to classify the Methanosarcina Lacustris from the other archaea.

4.3.4 GC Content

Figure 4.16 shows the GC counter on all the datasets, with the average, max, and min accuracy. Since this method has no randomness, there is no difference between the min and the max.

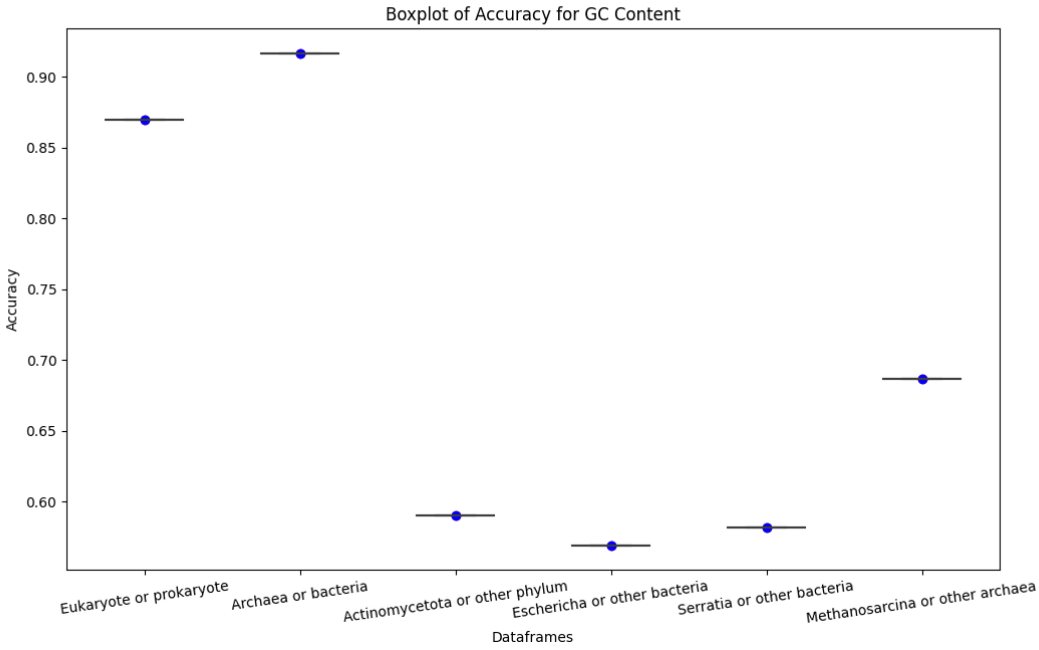

Figure 4.16: The average, max, and minimal accuracy of the GC content classifier, with input length of 1000, on the six different datasets

Figure 4.16 shows that only the datasets eukaryote or prokaryote and archaea or bacteria reach accuracy numbers over 0.85, and the closest is less than 0.7.

4.3.5 Convolutional Neural Network

Figure 4.15 shows the Convolutional Neural Network classifier on all the datasets, with the average, max, and min accuracy.

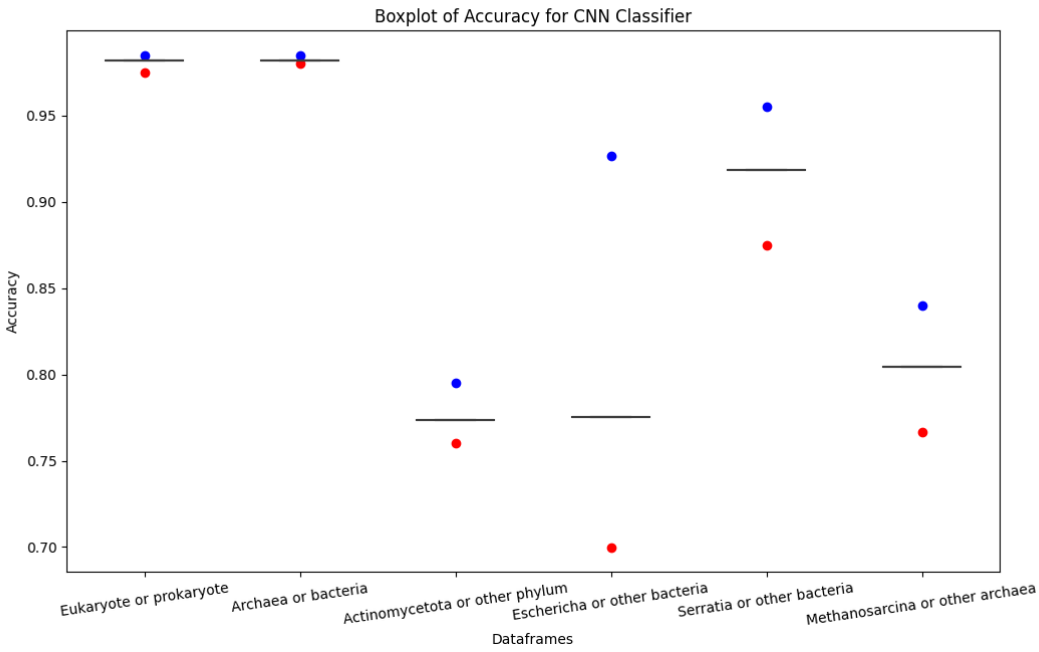

Figure 4.17: The average, max, and minimal accuracy of the Convolutional Neural Network classifier, with input length of 1000, on the six different datasets

Figure 4.17 shows that the CNN varies quite a bit between datasets, struggling to classify some of the datasets correctly. It also has a big variance in classifying Escheria Coli from the other bacteria.

Chapter 5

Discussion

In this section, this thesis's findings will be discussed based on the results in the previous chapter, using the questions asked in chapter one as groundwork.

5.1 The accuracy of the Tsetlin Machine

The accuracy results for the TM classifiers are very promising, at least the K-mer encoded one. It is very similar to Random Forest Classifier on all the datasets and beets the Convolutional Neural Network on multiple datasets. It seems to need more data than the Random Forest Classifier and the Convolutional Neural Network to get good results, but on 100 base pair input, it is about the same as the CNN and the RFC.

5.1.1 The direct encoded Tsetlin Machine

On the other hand, the directly encoded TM is not doing as well. It has way worse results than the other three machine learning methods, even worse than the GC counter on multiple datasets.

5.1.2 Direct coding vs kmer coding

The TM with direct coding is way worse at finding patterns in the data, rarely getting better than the GC content classifier. It seems this way of encoding the data makes it very hard for the TM to recognize features in the data. This is likely since the kmer way of encoding the data, the identical K-mers stay in the same place, so if the data is [0,0,1,0,0.....,0], the 3rd 1 means the same kmer in all the inputs, so if that specific kmer is a good indicator of it belonging to one particular class, that is a pattern the TM can easily make out, and finding relationships between the K-mers would also not be challenging. However, that would be far harder in the direct coding example. In the direct coding example, it would have to make a clause that can recognize multiple letters next to each other to identify one important kmer, and that does not take into account the relations between the K-mers. It doesn't even know when one letter begins, so that is also something it has to learn, that every 4 data inputs equate to one letter. The TM with direct encoding can probably only make out some straightforward patterns, such as the amount of GC in the input or something similar.

5.2 The speed of the Tsetlin Machine

The direct encoded TM is way worse than all the other methods, and there is probably more than one reason for this. One reason is that the input to the TM becomes larger the larger the

input is because the input is just letters in binary form. Its input is based on the length of the DNA string, so let's call that L, then the size of the input will be (1000,$4\dot{L}$) Where 1000 is the number of different inputs, which is the same always, and $4\dot{L}$ is four times the length L of the DNA string, since it is 4 bits in binary per letter. While on the k-mer encoded, the size of the input stays constant at (1000, 1024) where 1000 is again the number of inputs, and 1024 is the possible number of five letter combinations of the four different DNA bases $4^5 = 1024$ before it is split into testing and training data.
If we only compare the k-mer encoded Tsetlim machine to the other methods, it's much closer. For some reason, it takes much longer for an input length of 20 than the other, it is not clear why this is. The model is not quite as fast as the Random Forest Classifier and, of course, not as fast as the GC counter, but the longer the input length, the more it overtakes the CNN classifier on speed on higher input lengths. The TM is faster than the CNN.
This is all run locally on the CPU of a personal computer, so the result may have changed if it had been run on a GPU that could potentially parallelize the training process. Especially on the Convolutional Neural Network, there is a lot of literature on the increased speed of using a GPU over a CPU on neural networks[16][17]. Some work is being done on exploring the parallelization of the TM as well [18], but it hasn't been implemented yet. The focus has been on the time taken to train and classify this data instead of only the time taken to predict the classes based on the trained model. It would be interesting to have the models only timed on the prediction part after training and see if there is a big difference in the models.

5.3 Difference between datasets

It's also interesting how different datasets affect the accuracy. The three best classifiers, the K-mer encoded TM, the RFC, and the CNN, were around the same on The datasets: "eukaryote or prokaryote", "archaea or bacteria", or Serratia or other bacteria", but the CNN was significantly worse on "Actinomycetota or other phyla", "Escherichia Coli or other bacteria" and "Methanosarcina Lacustris or other archaea". It's also interesting that it's easier for the TM and RFC to classify some specific species from the others, but struggling classifying the one phylum from the others, likely this is a case of "overfitting" the one species, while it being harder to find features from all the species of a phylum.

5.4 Future Work

5.4.1 Real world data

One big thing that can be done to test out the TM further is to work on real-world data. The data used in this thesis is "perfect" generated reads and contigs. It would be interesting to see if the accuracy would be significantly worse. It could also be interesting to check on more types of data. This data is from the seabed. It would be interesting to see if data collected from somewhere else would generate different results or if it would be similar.

5.4.2 Tsetlin Machine variations

This thesis uses the vanilla TM, there is many different variants of the TM, the Convolutional Tsetlin machine [2], to simulate how the Convolutional Neural Network work, and the Coalessed Tsetlin Machine [19] to name a few. It's also possible to further play with the hyperparameters of the TM, for example, using weighted clauses [20] to decrease the number of needed clauses or drop clauses [21].

5.4.3 Data encoding

Playing more around with the encoding of data could also be something that can be worked on. There exist some variants of K-mer with gaps that can use larger K values without larger datasets [22]. It would be interesting to see if the TM could find features even easier from the data with different encoding.

Chapter 6

Conclusion

This thesis aims to evaluate the Tsetlin Machine's capabilities at taxonomically classifying pure genetic data and see if it compares to other standard methods in accuracy and speed. It has been compared to two other machine learning models, a Random Forest Classifier (RFC) and a convolutional Neural Network (CNN). It has also been compared with a baseline method of counting the amount of C and G in the DNA sequences. The input sequences given to the Tsetlin Machine (TM) have been encoded in two different ways: direct encoding and K-mer encoding. Different classes of taxonomic rank were tried on all the classifiers. All the models were compared using the same datasets, and the same classes.
The results for the accuracy of the TM with K-mer encoding are promising, reaching impressive results for relatively short input lengths. It is comparable to the RFC on every dataset, even beating it in some instances, and consistently better than the CNN.
The results on the time the TM with K-mer encoding took are quite a bit worse than the Random Forest Classifier. For short input lengths, it was also slower than the Convolutional Neural Network. On longer input lengths, it did catch up to the CNN and ended up being faster to train at input lengths over 2000. The directly encoded Tsetlin Machine does not produce promising results. It is less accurate and far slower than all the other machine learning models. In multiple instances, it performs worse than even the GC counter.
This is very promising for the TM with K-mer encoding, and shows that it has great potential in dealing with DNA classification tasks.

Bibliography

[1] Adrian Wheeldon et al. "Learning automata based energy-efficient AI hardware design for IoT applications". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2182 (Sept. 2020). Publisher: Royal Society, p. 20190593. DOI: 10.1098/rsta.2019.0593. URL: https://royalsocietypublishing.org/doi/full/10.1098/rsta.2019.0593 (visited on 05/13/2024).

[2] Ole-Christoffer Granmo et al. *The Convolutional Tsetlin Machine*. arXiv:1905.09688 [cs, stat]. Dec. 2019. DOI: 10.48550/arXiv.1905.09688. URL: http://arxiv.org/abs/1905.09688 (visited on 05/13/2024).

[3] Geir Thore Berge et al. *Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization with Medical Applications*. arXiv:1809.04547 [cs, stat]. Sept. 2018. DOI: 10.48550/arXiv.1809.04547. URL: http://arxiv.org/abs/1809.04547 (visited on 05/13/2024).

[4] Kristian Hovde Liland et al. "Tsetlin Machine in DNA sequence classification : Application to prokaryote gene prediction / A match made in silico". In: *2023 International Symposium on the Tsetlin Machine (ISTM)*. Aug. 2023, pp. 1–7. DOI: 10.1109/ISTM58889.2023.10454960. URL: https://ieeexplore.ieee.org/abstract/document/10454960 (visited on 04/22/2024).

[5] Ole-Christoffer Granmo. "The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic". en. In: ().

[6] K. Darshana Abeyrathna et al. *A Scheme for Continuous Input to the Tsetlin Machine with Applications to Forecasting Disease Outbreaks*. en. arXiv:1905.04199 [cs]. June 2019. URL: http://arxiv.org/abs/1905.04199 (visited on 01/15/2024).

[7] Olga Tarasyuk et al. "Systematic Search for Optimal Hyper-parameters of the Tsetlin Machine on MNIST Dataset". In: *2023 International Symposium on the Tsetlin Machine (ISTM)*. Aug. 2023, pp. 1–8. DOI: 10.1109/ISTM58889.2023.10454969. URL: https://ieeexplore.ieee.org/document/10454969 (visited on 04/22/2024).

[8] Barry de Ville. "Decision trees". en. In: *WIREs Computational Statistics* 5.6 (2013). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wics.1278, pp. 448–455. ISSN: 1939-0068. DOI: 10.1002/wics.1278. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.1278 (visited on 05/13/2024).

[9] Leo Breiman. "Random Forests". en. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324 (visited on 05/13/2024).

[10] SAGAR SHARMA. *What the Hell is Perceptron?* en. Oct. 2019. URL: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53 (visited on 05/13/2024).

[11] FRANCKE PEIXOTO. *A Simple overview of Multilayer Perceptron(MLP)*. en. Dec. 2020. URL: https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/ (visited on 05/13/2024).

[12] Jason Brownlee. *How to Choose an Activation Function for Deep Learning*. en-US. Jan. 2021. URL: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/ (visited on 05/03/2024).

[13] Frederik vom Lehn. *Understanding the Convolutional Filter Operation in CNN's*. en. Nov. 2023. URL: https://medium.com/advanced-deep-learning/cnn-operation-with-2-kernels-resulting-in-2-feature-mapsunderstanding-the-convolutional-filter-c4aad26cf32 (visited on 05/13/2024).

[14] *machine-learning-articles/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling.md at main · christianversloot/machine-learning-articles*. en. URL: https://github.com/christianversloot/machine-learning-articles/blob/main/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling.md (visited on 05/13/2024).

[15] *Terminology*. en. URL: https://bioinformaticsworkbook.org/introduction/dataTerminology.html (visited on 05/13/2024).

[16] Intisar Alkaabwi. "Comparison Between CPU and GPU for Parallel Implementation for a Neural Network Model Using Tensorflow and a Big Dataset". en. In: ().

[17] Dipesh Gyawali. *Comparative Analysis of CPU and GPU Profiling for Deep Learning Models*. en. arXiv:2309.02521 [cs]. Dec. 2023. URL: http://arxiv.org/abs/2309.02521 (visited on 05/13/2024).

[18] Anders Refsdal Olsen. "A Scalable Architecture for Parallel Execution of the Tsetlin Machine". eng. Accepted: 2019-09-25T12:40:07Z Publication Title: 89 p. MA thesis. Universitetet i Agder ; University of Agder, 2019. URL: https://uia.brage.unit.no/uia-xmlui/handle/11250/2618755 (visited on 05/13/2024).

[19] Sondre Glimsdal and Ole-Christoffer Granmo. *Coalesced Multi-Output Tsetlin Machines with Clause Sharing*. en. arXiv:2108.07594 [cs]. Aug. 2021. URL: http://arxiv.org/abs/2108.07594 (visited on 05/13/2024).

[20] Adrian Phoulady et al. *The Weighted Tsetlin Machine: Compressed Representations with Weighted Clauses*. arXiv:1911.12607 [cs, stat]. Jan. 2020. DOI: 10.48550/arXiv.1911.12607. URL: http://arxiv.org/abs/1911.12607 (visited on 05/14/2024).

[21] Jivitesh Sharma et al. *Drop Clause: Enhancing Performance, Interpretability and Robustness of the Tsetlin Machine*. arXiv:2105.14506 [cs]. Jan. 2022. DOI: 10.48550/arXiv.2105.14506. URL: http://arxiv.org/abs/2105.14506 (visited on 05/14/2024).

[22] Rong Wang, Yong Xu, and Bin Liu. "Recombination spot identification Based on gapped k-mers". en. In: *Scientific Reports* 6.1 (Mar. 2016). Publisher: Nature Publishing Group, p. 23934. ISSN: 2045-2322. DOI: 10.1038/srep23934. URL: https://www.nature.com/articles/srep23934 (visited on 05/13/2024).

Appendix A

Table of Python-packages

Python name	Version	Purpose of use
”numpy”	1.19.5	To calculate statistics
”Pandas”	1.1.5	To manipulate data frames
”tensorflow”	2.6.2	To create the Convolutional neural network
”scikit-learn”	0.24.2	To help preprocess data and create the Random forest model
”seaborn”	0.11.2	Plotting
”matplotlib”	3.3.4	Plotting
”Bio”	1.5.2	Open fasta files

Table A.1: The Python packages that are used to create the code and their respective uses.

The code I have used in this thesis can be fount at this link:Github Repository

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway