Bachelor's Thesis Computing Science

Radboud University Nijmegen

Weighted Negative Selection

Algorithm imitates life

Author:
Abel Lucassen
s4477464

First supervisor/assessor:
dr. Johannes Textor

Second supervisor:
Gijsbert Schröder, MSc

Second assessor:
prof. dr. Elena Marchiori

June 9, 2022

Abstract

Negative Selection is a classification algorithm based on the identically named mechanism in the human immune system. This algorithm classifies based on strict grouping into anomalous or non-anomalous parts, and does not make a more nuanced assessment than that. This means that any information about the degree of anomalousness of any given part of the data is lost. By incorporating weights based on the frequency of the matching parts, we want to improve the performance. We have shown that there are notable increases in performance at lower word matching lengths, but at higher word lengths this algorithm performs as well as or slightly worse than the unweighted approach. Even though this addition to negative selection did not completely have the desired effect on performance, the produced results and insights help with finding an approach that does.

Contents

Chapter 1

Introduction

Many people will have heard of artificial intelligence, and how neural networks are based on how the human brain and its neurons function. However, the human brain is not the only intelligent system in the human body. The immune system is capable of learning, memory and pattern recognition [5], and specifically the system of T-cell negative selection allows the immune system to also learn to generalize [12]. Just like how artificial neural networks are learning systems inspired by the workings of neurons and the human brain, are artificial immune systems learning systems inspired by the workings of the human immune system and its components.
Research into these artificial immune systems has two potential benefits. On the one hand, the immune system is a learning system that has evolved to classify in situations with noise and adversarial input, which could add insights to the field of artificial intelligence. On the other hand, immunologists can gain insights on how the immune system learns. One of the tasks that is interesting for both these fields of research is the task of anomaly detection. The task of anomaly detection is detecting anomalies with only knowledge of what is normal or non-anomalous. The immune system's solution for this is negative selection.
In negative selection, a set of classifiers is filtered based on whether they recognize the "self" and is removed if it does. This results in a set of classifiers which does not react with the "self", but if it does react with something else, it is likely that thing is anomalous. In the human immune system, this happens based on specific cells called T-cells and their ability to bind to other cells. In artificial immune systems, this translates to strings and functions that determine whether two strings match or bind. An artificial implementation of negative selection is simply and similarly called a negative selection algorithm. Such algorithms have been produced before, and it has been shown that a negative selection algorithm can perform competitively in respect to other algorithms on the task of anomaly detection [9].
However, the negative selection algorithm is very binary in its classification,

something is either anomalous or it is not. This is a very strict classification that loses potential information about how anomalous something is. For example, a measure of how anomalous something is can be based on how often it occurs in the non-anomalous training data. It has not been researched whether incorporating matching that uses values based on the frequency of occurring in the training data can lead to increased performance.
Intuitively, it does seem advantageous to distinguish between different strengths in matching for the negative selection algorithm, and it could help improve its performance. Therefore, our research question is: will incorporating weighted matching improve the performance of the negative selection algorithm?

Chapter 2

Preliminaries

In this chapter, we present some of the underlying theory used in the rest of the research. Because we are going to test and evaluate our algorithms on the task of anomaly detection, we will first discuss what this task is and how we can evaluate the effectiveness of algorithms on this task. Because of our use of sequential data and looking at small sequences of this data, we will discuss the basics of n-grams and how they are generated for use in the algorithms in this research. And finally, for better understanding of the general concepts, we will explain negative selection in the immune system and how this inspired and translates to the negative selection algorithm.

2.1 Anomaly Detection

We evaluate the performance of various algorithms on the task of anomaly detection. Anomaly detection is the task of detecting anomalous data from a data set containing both anomalous and non-anomalous data. While it seems similar to binary classification at first glance, the challenge in the task of anomaly detection is that only training data of the non-anomalous class is available, the algorithm does not train with any anomalous data.
Some examples tasks of anomaly detection are: recognizing a different language in a text, detecting malicious behaviour by programs of an antivirus program or the identification of foreign cells in the human body by the immune system. The latter of these is of interest in particular because we will be using the negative selection algorithm in this research, which is based on the similarly named process in the human immune system.
For measuring the performance of the anomaly detection task, we will use a metric called "AUROC" (Area Under Receiver Operating Characteristics). For this we will first have to plot an ROC curve, by plotting the TPR (True positive rate, also known as recall) values against the FPR (false positive rate) values. These values are gained by moving through each index of a

given ranked result list, which is in this case ranked on how anomalous data is perceived to be. At every index the true positives (TP), false positives (FP), false negatives (FN) and the true negatives (TN) up until that index are counted (If two results were given the same score they are considered as having an equal rank or index). Here true positives are data points with positive labels that are correctly classified as positive, false positives are data points with negative labels that are wrongly classified as positive, true negatives are data points with negative labels that are correctly classified as negative, and false negatives are data points with positive labels that were wrongly classified as negative. These values are used to calculate the TPR and FPR at a given index i as follows:

$$TPR_i = \frac{TP_i}{TP_i + FN_i}$$

$$FPR_i = \frac{FP_i}{FP_i + FN_i}$$

When you plot the TPR on the y-axis versus the FPR on the x-axis, you create an ROC curve. The ROC curve itself is a graphical tool that helps to show how well an algorithm is able to classify. AUROC is a concrete value derived from the ROC curve. Since the TPR and FPR are always values between 0 and 1 the ROC curve can at most have an AUROC of 1, which is also a perfect score, since it means it has correctly classified all the data points. If one were to randomly assign classification to all the data points, the AUROC would be around 0.5 which is the worst possible score. If the AUROC is much lower than 0.5 it probably means that your classifications are inverted – or something else is creating systematic issues.
AUROC is a valuable metric in this case since it incorporates multiple variables and also accounts for relative positions in the ranking of the test data, instead of just right and wrong. Since the algorithms used in this research produce a ranking of results, we will use this metric to evaluate them.

2.2 N-grams

N-grams are a way of representing parts of a data sequence. We test our implementations of various algorithms on a lot of sequential data, and will be using n-grams to represent parts of this data. N-grams are a sequence of n similar items, where the items are from a sequential source, like words or characters in a text or base pairs in DNA sequencing. N can have any value, but this is usually dependent on the problem it is used in. N-grams can be generated from a sequential data source in multiple ways. But in this research we will be using a sliding window approach.
In a sliding window approach, the first N items are copied starting from an

index, before moving the index by one. If there are not N items available in the remaining sequence, no further n-grams will be created. For example, if we take the input string "ABCDE" and N = 3, it would be transformed into the 3-grams: "ABC", "BCD", and "CDE".

2.3 Negative Selection

The negative selection algorithm is inspired by a similarly named process within the human immune system. Such an algorithm based on the human immune system can also be referred to as an artificial immune system. To understand the general principles of this negative selection algorithm, it is useful to understand the processes in the immune system it is based on and how it translates to a computer algorithm.
The human immune system consists of many complex parts. One of these parts is capable of producing cells that bind and recognize to specific cells using receptors. The immune system uses this interaction to find and destroy infected cells. But how do you get a set of receptors that can identify all types of infected cells, without wrongly identifying and destroying healthy cells?
The immune system uses negative selection to solve that problem. An illustration of negative selection can be seen in figure 2.1. In the thymus, T-cells (a type of white blood cell created in the bone marrow) mature and gain random T-cell receptors (TCRs). What a T-cell's TCRs bind to is different for each T-cell. This creates a large selection of T-cells with a large selection of TCRs. They are then, still in the thymus, shown a selection of cells that are native to the human body. Any of the newly created T-cells that bind or react to the 'normal' cells are destroyed, since this is unwanted behaviour. The remaining T-cells do not react to the body itself but could still react to cells it has not yet seen, like cells infected with a virus it has not seen before. If the set of TCRs is large enough, it should theoretically be able to recognize any anomalous cells without attacking its own body. In practice this is not the case, since many viruses and other attackers have ways of evading the immune system, like HIV (human immunodeficiency virus). However, the immune system is still often able to distinguish between anomalous cells and non-anomalous cells.
This ability to classify and distinguish is useful, and by mimicking the process we can create a classification algorithm. Such an algorithm is called a negative selection algorithm. In this algorithm, instead of using T-cells with receptors and observing whether they bind or not, we transcribe our sequential data to strings and perform string matching using some matching function on the n-grams generated from the strings. By mimicking a process from the immune system to create an algorithm, we can now use the

same principles for classification tasks on digital data. An example of how we can mimic the principle in a digital environment can be seen in figure 2.1.

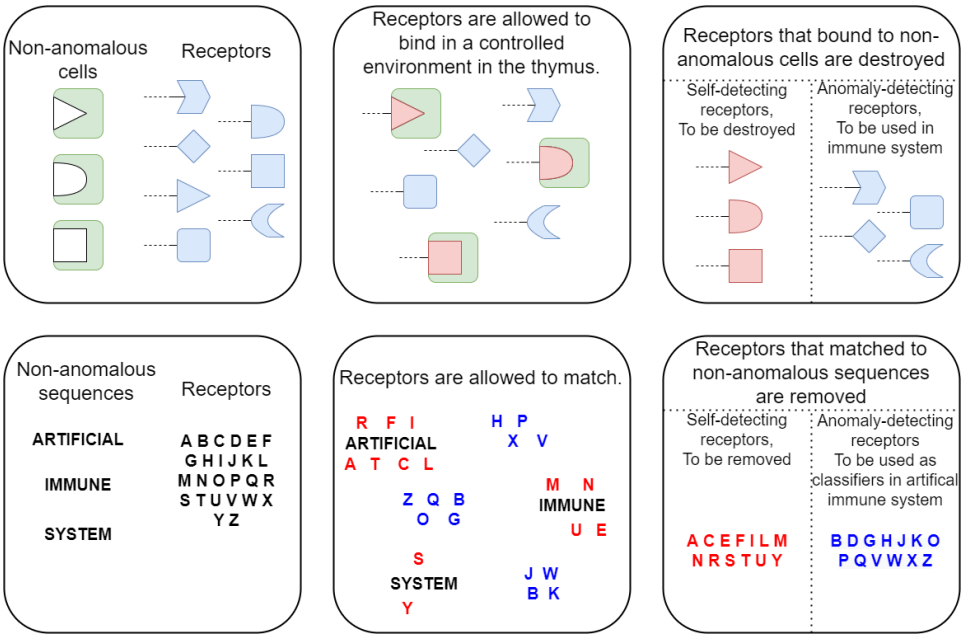

Figure 2.1: An illustration of how negative selection works in the human body, and an example of how we can mimic this process. On the left, we see the separate parts that are used in the process of negative selection. In the middle, we can see that these parts are allowed to bind or match depending on the specific environment. On the right, we see how the receptors that interacted with their respective non-anomalous parts are removed, so that only the receptors that are detecting anomalous parts remain.

Chapter 3

Methods

We compare the performance of a simple Naive Bayes classifier and two versions of a negative selection algorithm, each with two different matching functions. We test these algorithms with various n-gram lengths and different threshold values on the negative selection algorithms on 10 datasets. For each combination of variables and matching functions we will calculate the AUROC score to evaluate them.

3.1 Naive Bayes N-gram Classifier

One of the algorithms we implemented is an N-gram classifier. This algorithm will serve as a baseline and point of comparison to the other algorithms we will discuss later in this chapter. The N-gram classifier works by taking a set of string-like objects as input and transforming this into a dictionary of n-grams and how often they occurred in the input data.
When a string to be classified is then given, it is similarly converted into n-grams of the same size as the training data was. Each of these n-grams are then given a score that represents how anomalous these n-grams are, using the following formula:

$$n\text{–}gramscore = \frac{c_i + 1}{t + n}$$

where i represents a specific n-gram, c_i is the number of times that n-gram i is present in the dictionary, t is the total amount of n-grams in the training set and n is the total number of unique n-grams in the dictionary. We take the n-th root of the product of n-gram scores in a sequence, where n is the number of separate scores:

$$score(X) = \sqrt[n]{x_1 \times \ldots \times x_n}$$

where $X = (x_1, \ldots, x_n)$, a vector of anomaly scores of a sequence of n-grams. This operation is also known as the geometric mean and we use this

to remove the impact of length of the data on its final score. If we score an entire set of sequences, we can then sort them by their score and gain a ranking based on how anomalous a sequence is perceived to be, where lower scores are perceived to be more anomalous.

3.2 Negative Selection

The negative selection algorithm is based on the similarly named process in the human immune system, as discussed in the preliminaries. We use the algorithm of Textor et al. [10] in regard to using finite state automata for the creation of populations of classifiers, as will be discussed later.
In negative selection we determine a score based on how many 'receptors' in a trained 'repertoire' match with n-grams of a data sequence in the test set. Even though the classifiers and n-grams are both strings in the same language, we will call a set of characters from the data an n-gram, and a set of characters used for matching a receptor or classifier. Whether a receptor matches with an n-gram or not is determined by matching functions using a specific set of rules. To generate the trained 'repertoire', a set of all possible receptors is created and the receptors that bind to the n-grams of the training set are removed. The trained repertoire can no longer react with the 'self', but can still react to everything else. Now we should be capable of distinguishing from non-anomalous and anomalous data sequences by how often receptors in the generated 'repertoire' bind to them.
The algorithm starts by training on non-anomalous data. Let N be a positive integer value chosen to represent the size of n-grams to be used in training, matching and classifying. Then we create a population of classifiers that contains all possible n-grams. These classifiers are our receptors, the population is our repertoire and in our case the items used will be characters. This means that every classifier is a string and every population is simply a set containing these classifiers. For example, if we would make an initial population of possible strings of lower case characters with a word length of three, we would have a population that consists of 26^3 classifiers. This means that there will be potential limitations due to memory or runtime constraints when working with larger alphabets or higher word lengths.
We also create a "self-reactive" population of classifiers. This is realized by creating a finite state automata (FSA) that only accept the words that matches to an n-gram, for all n-grams in the training data. The FSAs and operations on them are implemented using the OpenFST library [1]. We will explain the exact workings of the matching functions and how the FSAs are constructed later on.
Now that we have a total population and a self-reactive population, we can create an anomaly-detecting population by simply using the total population minus the intersection of the total and self-reactive populations (since

the total population contains all possible classifiers, this is equal to the total population minus the self-reactive population). The anomaly-detecting population only contains classifiers that do not match with any word or n-gram in the training set, but are still able to react with anomalies.
Now that we have an anomaly-detecting population (a repertoire of classifiers that do not react with the 'self', but can still detect anomalies), we can start classifying. For every data sequence we want to test, we first split that sequence into n-grams using the sliding window approach. Then for each n-gram in this sequence we create a population of classifiers similar to how we made the self-reactive populations. We then take the intersection of this population and the anomaly-detecting population we created earlier to create a population of matching classifiers for this word. The amount of classifiers in the intersected population is the number of matches of this n-gram in the anomaly-detecting population, which we will note down. Once we have done this for each n-gram in the sequence, we take the arithmetic mean of the amount of matches and this will also act as the score for this sequence. A sequence with a higher score is therefore one that is considered more anomalous to the original training set.

3.2.1 Matching functions

The matching function in the negative selection algorithm is important to determine which n-grams are considered a match to a specific classifier. A simple choice would be to only consider an n-gram as a match if it is exactly the same, but we want to have more options than just that. The two different matching functions we will be using are r-contiguous and Hamming distance. Both of these functions take as input: the n-gram of which we want to find the matching words and a threshold value, which is an integer value of at least 1 and at most the length of the input n-gram.
In the r-contiguous function, a word is considered a match if a contiguous sequence of at least the threshold value in size is equal to the input n-gram. Note that the matching sequence needs to match on the same indices. For example, if we have the alphabet consisting of A and B and our input N-gram is "AAA" and a threshold value of 2 the matching n-grams would be: "AAA", "AAB" and "BAA".
The Hamming distance function is similar to the r-contiguous function, except that we do not require the matching parts to be a single sequence. To be a match for the input n-gram, we only need a number of identical items on the same index equal or higher than the threshold value. Using the same example as with the r-contiguous function we would gain the matching words: "AAA", "AAB", "BAA" and "ABA".
We find the matches to a n-gram by creating an FSA according to the chosen matching function, word length and threshold values. This FSA accepts only those matches and can then be used to generate the population of matches.

This is faster than creating a large list of strings directly.
For intuition, it helps to see the states of the FSA as if they were laid out in a rectangular pattern, where the rows translate to the number of "hits" (a successful match of the item in that index) so far. Secondly the columns translate to the position in the word. When building the FSA, edges are added for all items in the alphabet, with a "hit" translating to an edge that goes down by one and a "miss" translating to either just going one node to the right in the case of Hamming or resetting back to the top row in the case of r-contiguous. For reachable edges in the bottom row that are not in the last column, an edge will be created for the full alphabet to go one to the right. The bottom most right most node is always the only finite state in the FSA. In figure 3.1 you can see the respective FSAs for r-contiguous and Hamming as used in the examples above.

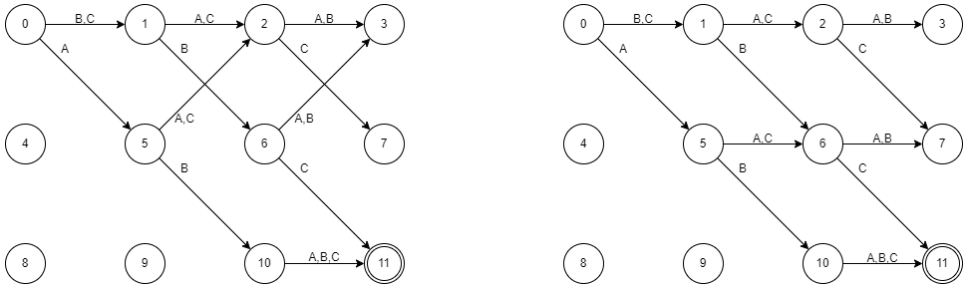

Figure 3.1: Two examples of how FSAs can be used to create all matching receptors of an n-gram and a threshold value: Left is r-contiguous, Right is Hamming

3.2.2 Weighted Negative Selection

We made a slightly modified version of the negative selection algorithm that incorporates weighted matching. For weighed matching we can make use of WFSAs, FSAs where each edge has a weight. The weight of a classifier that is accepted by an WFSA is equal to the product of the weights of the edges along the path from the initial to the accepting state. We generally give all edges a weight of 1 so that different values for word length or threshold influence the score. If we want to change the weight of a classifier we can do so by giving the accepting state an extra weight.
Classifiers in the population of all possible classifiers are created with a weight of 1, Classifiers in the self-reactive population are created with a weight equal to the negated frequency (a number between -1 and 0) of the total amount of n-grams in the training set. Since we do not have a function that can subtract the weights in the self reactive population from the total population, we have to add the negated self-reactive population to the total population instead. When both sets contain the same classifier, the weights

of these classifiers are added up (and only one of these classifier remains). The idea behind this approach is that the algorithm also takes into account how often a certain n-gram appears in the training set. This should make it so that the scores better represent how anomalous a sequence is.

3.3 Datasets

In this research we look at the performance on ten datasets from the work of Chandola et al. [2] five of these datasets are protein family datasets while the other five are intrusion detection datasets. The 5 protein family datasets are HCV, NAD, RUB, RVP, and TET. Each of these datasets consists of a training set, containing amino acid sequences of the respective protein family, and three labeled testsets of differing size and anomaly ratios. Anomalies in these testsets are amino acid sequences from other protein families.
The five intrusion detection datasets are UNIX system calls translated to a sequence of characters. These datasets also consist of a training set and three different testsets. The anomalies in these sets are system calls made by processes that are compromised from an attack.

3.4 Testing setup

We will evaluate the results of the Naive Bayes classifier and the two implementations of our negative selection with two matching functions each. We will do this by training and classifying the algorithms on 10 different datasets and different values of the parameters word length and threshold. An overview on what algorithms are run with which variables can be found in table 3.1. Some additional results for the HCV and snc-cert en snd-unm have also been produced and are discussed in the results chapter.

Table 3.1: Overview of variables tested. There were more planned results, but due to runtime constraints not everything has completed.

Algorithms	Matching Function	Word Length N	Threshold R
Protein family datasets			
Naive Bayes	-	1-6	-
Negative Selection	R-contiguous	1-6	R=N
Negative Selection	Hamming	1-4	R=N, ..., R=N-3
Weighted Neg. Sel.	R-contiguous	1-6	R=N
Weighted Neg. Sel.	Hamming	1-4	R=N, ..., R=N-3
UNIX system calls			
Naive Bayes	-	1-6	-
Negative Selection	R-contiguous	1-6	R=N
Weighted Neg. Sel.	R-contiguous	1-6	R=N

Chapter 4

Results

In this chapter, we will show and discuss the results of the performed tests. We have run a Naive Bayes classifier and two versions of a negative selection algorithm, each with two matching functions on 10 data sets. Each run has produced a list of scores on test data, of which we have taken the AUROC to evaluate their effectiveness and the effects of our tested variables. All results can be found in appendix A, but we will summarize the most important findings here.

4.1 General Observations

First, we will look at the specific case of exact matching (where the threshold value is equal to the word length). Overall, one of the most important variables here is word length. From the results we have obtained, we can see that exact matching does not perform as well on lower word lengths compared to higher word lengths, as can be seen in figure 4.1. The three bsm datasets are a good example that show a general increasing trend with word length. However, there are also datasets in which an optimum word length is reached at word length 4 or 5, like the HCV and TET datasets, as can be seen in figure 4.1. Even though an algorithm using higher word lengths could learn more complex patterns, we hypothesise this is only true as long as there is enough data that these patterns can be found, and that the optimal word length is not only dependent on the data type and characteristics but also the amount of data available for learning.
An unexpected result is that we have multiple AUROC values that are considerably under 0.5. This means that in those runs, the algorithm might be actively learning wrong or counterproductive information. This seems to happen in runs with low word lengths on all three bsm datasets, and it only occurs in the algorithms of the Naive Bayes classifier and the weighted negative selection, not in the regular negative selection. From this, we hypoth-

esise it is related to the frequency of certain characters from the alphabet in those datasets. Either there are characters in the alphabet that often occur in the training data and are considered to be very non-anomalous, even though these are also common in anomalous data, or the opposite where characters are not occurring often in the training data also do not occur often in anomalous data. This could result in the algorithm to be too strict or not strict enough to certain characters, resulting in a biased scoring against a certain class.

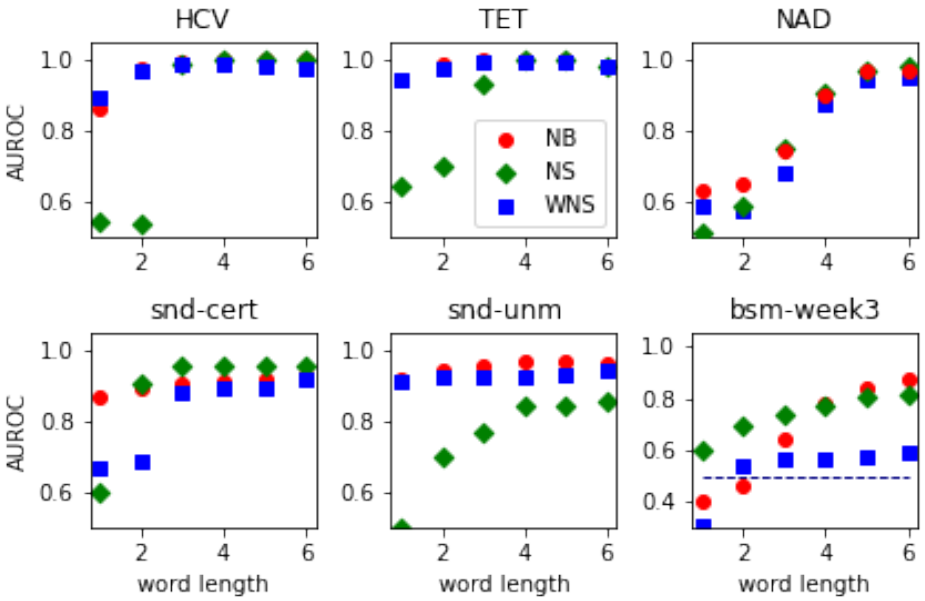

Figure 4.1: Influence of word length N on performance in cases of r-contiguous where $r = N$. Here we see the general increase of performance (AUROC) with increasing word lengths on all three algorithms. These are three of the protein family datasets (HCV, TET and NAD) and three of the UNIX system call datasets (snd-cert, snd-unm and bsm-week3) chosen to give a representative selection of the produced data. The axis on the bsm-week3 plot has been altered to show all data points, and a line is plotted at an AUROC value of 0.5 to demonstrate that this is an unexpected result. We can see the increase in score with increase in word length all in graphs, except for the HCV and TET plots, in which it seems an optimum word length is achieved at a word length of around 4 or 5.

4.2 Performance of weighted against unweighted selection differs per dataset

In figure 4.1 we can see that with few exceptions the weighted version of negative selection performs roughly equal or (slightly) worse than the base version of negative selection at word lengths of 3 and higher when using exact matching. However, we can also see that at lower word lengths on all protein family datasets, there are significant result gains when using the weighted negative selection. The algorithm performs worse on weighted negative selection on lower word lengths on the bsm datasets, as mentioned earlier. On snd-unm dataset, the weighted negative selection performs notably better at all word lengths. On the snd-cert dataset however, the algorithm performs somewhat worse when using weighted negative selection. Most of these examples are visible in figure 4.1, all other results can be found in appendix A.
From this, it seems that the effect of the weighted negative selection versus the base negative selection is heavily dependent on the data type, alphabet size or word lengths. The bsm datasets have the largest alphabets of the datasets by far, but at this point it is unclear whether this is the specific reason for its performance or not.

4.3 Threshold value influences performance more than word length

In figure 4.2 we see examples of Mamming runs on the HCV dataset, in the figures the scores are coloured according to their threshold value. One of the most noticeable results when looking at this figure, is that on runs using a threshold value r lower than the word length n, the AUROCs seem to be more dependent on the value of r than on the value of n.
We saw from our results that when running with r=n, that the unweighted algorithm performed best on the HCV dataset at n=5 and the weighted algorithm performed best at n=4. In figure 4.2 we see that regardless of word length, the runs using those optimal word lengths as threshold value perform the best or very close to it. For example, we see that for unweighted HCV hamming at word length and threshold value of 3 the algorithm performs better than any algorithm using word lengths of 2 or lower. For higher values of word length, the performance for algorithms using a threshold value of 3 remains comparable.
It seems that at higher values of word length, it is no longer optimal to have the threshold value be equal to the word length. But we would need further testing at higher word lengths to see if this trend continues or that there is an optimal threshold value, similar to how some datasets had an optimal

word length.

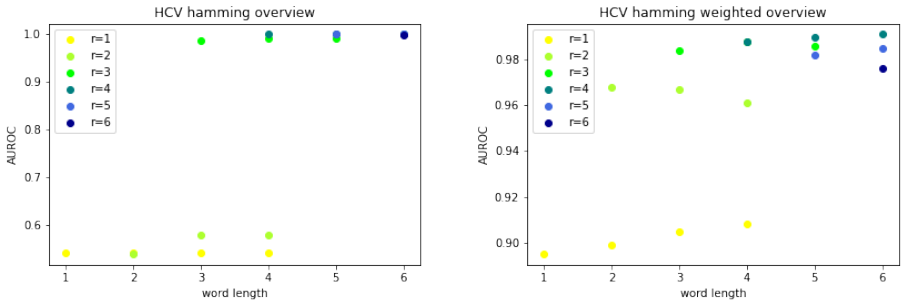

Figure 4.2: Here we see the influence of threshold value on performance. Runs with the same threshold value have the same colour to illustrate the strong relation between threshold value and performance. As we can see for the dataset of HCV when using hamming matching for both weighted and unweighted negative selection, is that word length influences the AUROC results less than the threshold value. The y-axes (AUROC) are scaled differently to improve readability.

4.4 Future research

We have seen that weighted negative selection can perform better than the base version in certain situations. However, there are many situations where this has not been the case. In future research, possible avenues of approach could be attempting to improve weighted negative selection by determining what causes the reduced performance of weighted negative selection, especially in the case of the bsm datasets. Testing different ways of incorporating weight or changing how much the weight influences the scoring might result in an improved version of the tested algorithms with more positives and/or fewer drawbacks. For example, by reducing the impact of weights by a large factor, you would get a performance closer to unweighted negative selection, except in cases when negative selection would be scoring all data equally, like in the case of lower word lengths, in which it would likely only be an improvement.
It could also be useful to continue research on this subject with a larger parameter space, as it could give more definitive answers on some results and hypotheses discussed earlier. Specific information that this could produce would for example be if every dataset has an optimal word length and if there are correlations between such an optimal word length and the size of the alphabet and the training sets. It could also help predict what influences

whether the weighted version performs better or worse on any given dataset. Besides a larger parameter space, evaluating more data types like languages and other approaches for generating n-grams like chunking instead of a sliding window could help to understand, and in turn improve the performance of weighted negative selection. Especially with the latter, it could be interesting to see if the relation between the threshold and performance stays the same for value of threshold that are lower than the word length.
Another line of research is to look at how much our understanding of the immune system and how much our understanding of artificial immune system can influence one another. By looking at how the immune system handles some of the issues we are facing we could try to adapt those solutions to our algorithms, and conversely if we notice a certain effect in our algorithms, this could inspire new hypotheses on how the immune system deals with certain issues or challenges. For example, we showed in our research that weighted negative selection had significant impact on the results when looking at lower word lengths. Maybe since we have found this to be a possible solution for situations where we have limited information, the human immune system has evolved to handle this problem similarly.

Chapter 5

Related Work

Artificial intelligence and the immune system initially seem like two different concepts that do not overlap much. The combination of artificial intelligence and the immune system is however not that new or niche. For example, as early as 1986, Farmer et al. [5] recognized that the immune system is capable of learning, memorizing and recognizing patterns and that this process was able to be simulated on a computer.
In 1994, Forrest et al. [6] researched the problem of discriminating between self and nonself, specifically in the context of protecting computer systems. They researched a solution based on the generation and behaviour of T cells in the immune system and show how this approach could be used to solve the problem of computer system protection.
Eventually however, criticism about the efficiency and scalability of negative selection and other algorithms based on the immune systems were published. Kim et al. [7] researched an artificial immune system for network intrusion detection and concluded: "the computation time needed to generate a sufficient number of detectors is completely impractical". More research on the complexity of the negative selection algorithm has been done by Stribor et al. [8] and Timmis et al. [11] with similar concerns, and the latter of these claims that the problem of generating detectors in a negative selection algorithm is equivalent to an NP-complete problem. A property that would confirm that the generation of detectors for negative selection is not practical.
Elberfield et al. [3] [4] have instead shown that negative selection can run in polynomial time instead of exponential time. They have proven that if an automata can be created in polynomial time, the detectors can be created in polynomial time as well.
This approach has been implemented and evaluated by Textor [9]. They compare the r-chunk and r-contiguous detectors versus various other techniques on 14 datasets from real-world sources and found that these methods performed as well or better. Textor et al. [10] continue this line of research

with more detectors on various related algorithmic tasks.
We are building on this approach of the negative selection algorithm by researching weighted matching and taking into account frequency in the matching and scoring of the algorithm.

Chapter 6

Conclusion

We have implemented negative selection with weights and tested how it performed against the normal unweighted variant. We have shown that weighted negative selection generally performs similarly or slightly worse than the unweighted version. It can however improve performance when using lower word lengths. In the future, we could evaluate whether weighted negative selection can be improved by increasing or decreasing how much the weights influence the final score, or how weighted negative selection performs with different matching functions or using a chunking approach instead of a sliding window for generating n-grams.

Bibliography

[1] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. Openfst: A general and efficient weighted finite-state transducer library. In *International Conference on Implementation and Application of Automata.*, pages 11–23. CIAA, 2007.

[2] V. Chandola, V. Mithal, and V. Kumar. A comparative evaluation of anomaly detection techniques for sequence data. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, page 743–748. ICDM, 2008.

[3] M. Elberfeld and J. Textor. Efficient algorithms for string-based negative selection. In *Proceedings of the 8th International Conference on Artificial Immune Systems*, page 109–121. ICARIS, 2009.

[4] M. Elberfeld and J. Textor. Negative selection algorithms on strings with efficient training and linear-time classification. In *Theoretical Computer Science 412*, page 534–542, 2011.

[5] J. Farmer, N. Packard, and A. Perelson. The immune system, adaptation, and machine learning. In *Physica D: Nonlinear Phenomena, Proceedings of the Fifth Annual International Conference*, pages 187–204, 1986.

[6] S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, page 202–212. IEEE Computer Society Press, 1994.

[7] J. Kim and P.J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 1330–1337. GECCO, 2001.

[8] T. Stibor, J. Timmis, and C. Eckert. The link between r-contiguous detectors and k-cnf satisfiability. In *Proceedings of the Congress on Evolutionary Computation*, page 491–498. IEEE Press, 2006.

[9] J. Textor. A comparative study of negative selection based anomaly detection in sequence data. In *Artificial Immune Systems*, pages 28–41. ICARIS, 2012.

[10] J. Textor, K. Dannenberg, and M. Liskiewicz. A generic finite automata based approach to implementing lymphocyte repertoire models. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, page 129–136. GECCO, 2014.

[11] J. Timmis, A. Hone, T. Stibor, and E. Clark. Theoretical advances in artificial immune systems. In *Theoretical Computer Science 403*, pages 11–32, 2008.

[12] I. Wortel, C. Keşmir, R. J. de Boer, J. N. Mandl, and J. Textor. Is t cell negative selection a learning algorithm? In *Cells 9(3)*, page 690, 2020.

Appendix A

Result Tables

We have ordered our results in six tables, corresponding to the word length used for those runs. In each table we will have one column for the results of the naive bayes classifier, one for the base version of the negative selection algorithm, and one for the weighted negative selection algorithm. In tables starting from word length 2, we will also be adding columns for the results of both versions of negative selection that use a threshold value lower than the word value. For some word lengths we have tested more threshold values, but we have not tested for values that are more than 3 lower than the word length, both because of runtime concerns and that initial testing showed that these results did not perform competitively.

In these tables there are a number of missing results, this is because many of these runs require an unexpectedly long runtime to complete. Run times for the negative selection algorithms increase not only with word length, but also when the threshold value is closer to half the word length. We have tried to produce as many results as possible while also producing a set of representative results that should be enough to show the differences between algorithms and variables.

Table A.1: Results for word length 1

Dataset	NB	NS	WNS
HCV	0.862	0.540	0.895
NAD	0.626	0.510	0.583
RUB	0.309	0.500	0.416
RVP	0.898	0.500	0.988
TET	0.941	0.640	0.945
snd-cert	0.871	0.599	0.668
snd-unm	0.919	0.500	0.915
bsm-week1	0.308	0.500	0.353
bsm-week2	0.426	0.570	0.321
bsm-week3	0.402	0.599	0.310

Table A.2: Results for word length 2

Dataset	NB	NS	WNS	NSH-1	WNSH-1
HCV	0.972	0.538	0.968	0.540	0.899
NAD	0.650	0.587	0.574	0.510	0.578
RUB	0.793	0.498	0.774	0.500	0.423
RVP	0.989	0.627	0.987	0.500	0.988
TET	0.986	0.697	0.971	0.640	0.937
snd-cert	0.894	0.908	0.689	—	—
snd-unm	0.947	0.699	0.926	—	—
bsm-week1	0.319	0.497	0.526	—	—
bsm-week2	0.415	0.627	0.510	—	—
bsm-week3	0.464	0.697	0.537	—	—

Table A.3: Results for word length 3

Dataset	NB	NS	WNS	NSH-2	NSH-1	WNSH-2	WNSH-1
HCV	0.995	0.987	0.984	0.578	0.540	0.967	0.905
NAD	0.740	0.746	0.678	0.605	0.510	0.594	0.589
RUB	0.989	0.989	0.922	0.498	0.500	0.733	0.442
RVP	0.997	0.996	0.992	0.644	0.500	0.991	0.988
TET	0.996	0.930	0.991	0.717	0.640	0.963	0.931
snd-cert	0.910	0.957	0.881	—	—	—	—
snd-unm	0.960	0.768	0.927	—	—	—	—
bsm-week1	0.414	0.495	0.538	—	—	—	—
bsm-week2	0.474	0.656	0.485	—	—	—	—
bsm-week3	0.640	0.736	0.564	—	—	—	—

Table A.4: Results for word length 4

Dataset	NB	NS	WNS	NSH3	NSH2	NSH1	WNSH3	WNSH2	WNSH1
HCV	0.998	0.999	0.988	0.990	0.578	0.540	0.988	0.961	0.908
NAD	0.901	0.902	0.875	0.746	0.605	0.510	0.644	0.604	0.592
RUB	0.997	0.997	0.994	0.990	0.498	0.500	0.890	0.709	0.451
RVP	0.998	0.998	0.994	0.996	0.751	0.500	0.995	0.991	0.988
TET	0.998	0.997	0.995	0.960	0.736	0.640	0.989	0.955	0.926
snd-cert	0.917	0.957	0.894	—	—	—	—	—	—
snd-unm	0.970	0.847	0.928	—	—	—	—	—	—
bsm-week1	0.554	0.643	0.566	—	—	—	—	—	—
bsm-week2	0.614	0.705	0.460	—	—	—	—	—	—
bsm-week3	0.779	0.775	0.566	—	—	—	—	—	—

Table A.5: Results for word length 5

Dataset	NB	NS	WNS	NSH4	NSH3	NSH2	WNSH4	WNSH3	WNSH2
HCV	0.999	0.999	0.982	0.999	0.990	—	0.990	0.986	—
NAD	0.966	0.966	0.941	—	—	—	—	—	—
RUB	0.996	0.997	0.996	—	—	—	—	—	—
RVP	0.997	0.997	0.996	—	—	—	—	—	—
TET	0.996	0.996	0.995	—	—	—	—	—	—
snd-cert	0.922	0.956	0.894	—	—	—	—	—	—
snd-unm	0.970	0.848	0.934	—	—	—	—	—	—
bsm-week1	0.584	0.642	0.604	—	—	—	—	—	—
bsm-week2	0.685	0.745	0.419	—	—	—	—	—	—
bsm-week3	0.840	0.805	0.575	—	—	—	—	—	—

Table A.6: Results for word length 6

Dataset	NB	NS	WNS	NSH5	NSH4	NSH3	WNSH5	WNSH4	WNSH3
HCV	0.997	0.998	0.976	0.999	0.999	—	0.985	0.991	—
NAD	0.969	0.978	0.948	—	—	—	—	—	—
RUB	0.998	0.997	0.996	—	—	—	—	—	—
RVP	0.997	0.997	0.995	—	—	—	—	—	—
TET	0.980	0.980	0.977	—	—	—	—	—	—
snd-cert	0.935	0.956	0.919	—	—	—	—	—	—
snd-unm	0.967	0.857	0.944	—	—	—	—	—	—
bsm-week1	0.634	0.642	0.611	—	—	—	—	—	—
bsm-week2	0.726	0.774	0.410	—	—	—	—	—	—
bsm-week3	0.875	0.815	0.593	—	—	—	—	—	—

Appendix B

Code

Code can be found at:
https://gitlab.computational-immunology.org/alucassen/abel-bachelor-thesis-ngram-classifier
There are three python files for naive bayes, negative selection and weighted negative selection respectively. These files work at least with python version 3.6.9 and 3.9.7. More instructions can be found in the readme file in the repository.