Assignment 2

Anomaly Detection using Negative Selection Algorithms

Deadline February 24th, 2025

Handout for the *Natural Computing* lecture, February 3rd, 2025

TA for this assignment: Jan Schering, Koert Schreurs

Objectives of This Exercise

1. Implement the Receiver Operating Curve (ROC) visualization and the Area Under the Curve (AUC) metric.
2. Analyze a negative selection algorithms's performance using ROC and AUC.
3. Apply a negative selection algorithm to a real-world dataset.
4. Design an analysis using negative selection on a real-world dataset
5. Substantiate your design decisions in a report describing your process.

In this assignment, you will apply the negative selection algorithm to anomaly detection in sequence data. The assignment consists of two parts: in the first, the algorithm implementation is explained using an artificial toy problem, in the second, you will apply this implementation to a more realistic dataset from the domain of network security.

This assignment description contains some introductory explanation and text. You are not expected to respond to these parts in your report. Respond only to those parts that are labelled as **your task**.

1 Using the Negative Selection Algorithm

For this assignment, you can work with the author's implementation of a negative selection algorithm. This implementation is written in Java, which you need to have installed. The program itself and the data files needed for this assignment can be found in `negative-selection.zip` on Brightspace.

Familiarize yourself with the implementation

Download and unpack `negative-selection.zip`. This will create a folder `negative-selection`. Open a terminal and change to that folder. Run the negative selection program using:

```
java -jar negsel2.jar
```

This will show a list of possible options.

Classifying languages

In this toy example, you will train an artificial T cell repertoire using fixed-length strings taken from the (English) book *Moby Dick*. Then, you will expose this repertoire to both English strings and strings from another language, which will be the "anomalies". If the algorithm is tuned well, it should be able to distinguish English and non-English strings. We start by training our repertoire on English strings. Run the following command:

```
java -jar negsel2.jar -alphabet file://english.train -self english.train -n 10 -r 4 -c
```

This will build a repertoire containing all patterns of length 10 (switch -n) that do not share any contiguous substring of length more than 4 (switch -r) with any string in the input set `english.train` [1] (where one string per line is given). It will then read input strings line by line from standard input. For each line, it will count (-c) the number of patterns in the repertoire that match this line. If the -c switch is omitted, it will instead output the length of the longest contiguous string shared by the input line and any pattern in the repertoire. Lastly, the -alphabet switch determines the basis alphabet for strings and patterns to be the set of unique characters found in the file given as the argument. The default behaviour is to use all characters found in the input file, so the switch can be safely omitted in this case. (It will be relevant in the next exercise, however, and is therefore introduced here.)

When the program expects your input, type in the following line:

```
call_me_is
```

This will output the number 0, because this string is the first line from the input dataset. Next input the line

```
fall_me_is
```

As we can see, this string can be matched by a vast number of patterns. Such huge numbers can be unwieldy to work with, so there is an option that lets us output each number x. as $\log_2(1+x)$ instead. Test this feature by ending the current session (by typing CTRL-D to end the input or CTRL-C to abort the program) and restart the program with the -l switch:

```
java -jar negsel2.jar -self english.train -n 10 -r 4 -c -l
```

Now, the output for `call_me_is` is still 0, but the output for `fall_me_is` is about 28.

Using input redirection, we can now determine the number of matching patterns for each string in the files `english.test` and `tagalog.test`. For example, on Linux, Mac or cygwin platforms, we could use a small inline `awk` script to compute the average number of matching patterns for both files:

```
java -jar negsel2.jar -self english.train -n 10 -r 4 -c -l < english.test | awk
    '{n+=$1}END{print n/NR}'
java -jar negsel2.jar -self english.train -n 10 -r 4 -c -l < tagalog.test | awk
    '{n+=$1}END{print n/NR}'
```

As we can see here, the average number of matching patterns is somewhat less than 2-fold higher for Tagalog strings than for English strings, so some learning does seem to occur.

To sum up, this program works in the following way after training the repertoire:

```
Input (string): The pattern to check for anomalies (e.g. fall_me_is)
Output (int): The number of detectors in the repertoire that match the input string
```

Your task

1. Compute the area under the receiver operating characteristic curve (AUC [1][2]) to quantify how well the negative selection algorithm with parameters $n = 10$ and different values of r ($r = 1$ until $r = 9$) discriminates individual English strings from Tagalog strings by using the files `english.train` for training and `english.test` as well as `tagalog.test` for testing. Which value of r belongs to what AUC in figure 1? (Use this to check your implementation) [2]

[1] In the lecture, we called these patterns *r-contiguous detectors*, and here we have $r = 4$.

[2] You compute the AUC curve by merging the two test string sets, and then sorting them by the "anomaly score" that is given to them by the negative selection algorithm. Then, for each possible cut-off score (that is, each distinct value in this list), you compute the sensitivity (percentage of "anomalous" strings higher than that score) and the specificity (percentage of "normal" strings lower than that score). Then you generate the AUC curve from those values. Or, simply use an R package such as AUC.

Tips: be careful with how you handle multiple rows with the same anomaly score, always include the points $(0,0)$ and $(1,1)$, and be careful not to swap the axes.

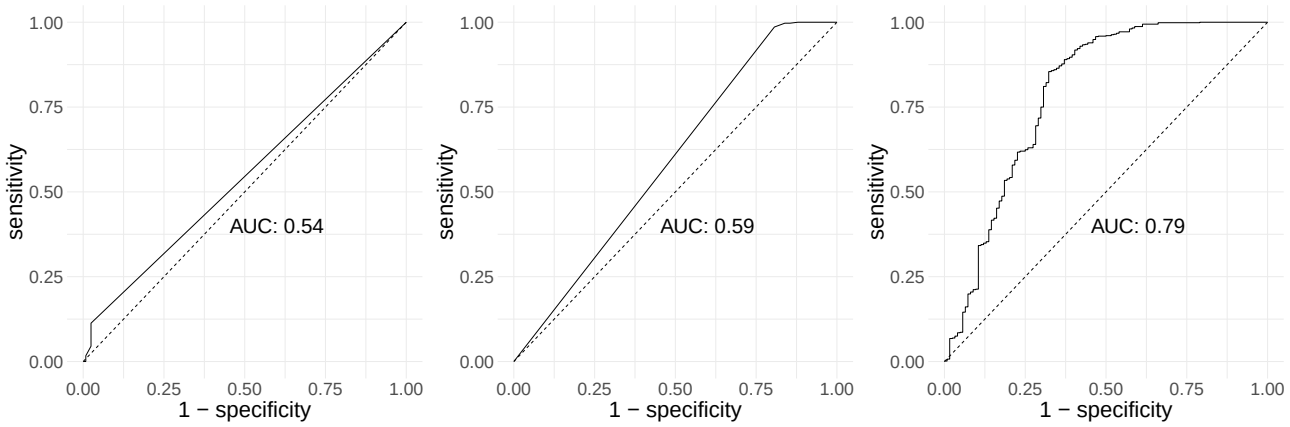

Figure 1: Receiver operated curves with AUCs for negative selection discriminating between English and Tagalog, with three different values for the parameter r. Task 1 (above) is to find out which values of r belong to which of these three figures.

2. How does the AUC change when you modify the parameter r? Specifically, what behaviour do you observe at $r = 1$ and $r = 9$ and how can you explain this behaviour? Which value of r leads to the best discrimination?
3. The folder `lang` contains strings from 4 other languages. Which languages can be best discriminated from English using the negative selection algorithm, and for which is this most difficult? Explain your findings.
4. We train the repertoire on strings of a certain length, but we could in theory input strings of other lengths. Explain in your own words the issue with providing strings that are too long/short.

2 Intrusion Detection for Unix Processes

We will now apply negative selection to a more interesting problem: intrusion detection on a Unix computer system.

Unix processes communicate with the kernel using system calls (such as `open`, `read`, `write` etc). Tools such as `strace` can be used to monitor the system calls made by a running process. Anomalous system call sequences (such as a normally silent process suddenly opening hundreds of files) could indicate an exploit.

In this exercise, we will work with such data, which is stored in the folder `syscalls` and has been collected many years ago in the group of Stephanie Forrest at the University of New Mexico[3]. This data has a similar structure to the language data from the previous exercise:

- Files ending with `.train` contain training sequences. All these are system call sequences recorded during normal behaviour of a process such as `sendmail`.
- Files ending with `.test` contain system call sequences representing both normal and anomalous behaviour.
- Each `.test` file has a corresponding `.labels` file that contains a 0 for a normal sequence and a 1 for an anomalous sequence.
- Finally, the `.alpha` file contains the alphabet used to encode the system calls – each symbol represents a different system call.

There are two subfolders in the folder `syscalls`, each containing the aforementioned files.

Your task: using negative selection, create a classifier that detects the anomalous sequences in the system calls datasets as well as possible! Perform an AUC analysis to evaluate the quality of your classification.

There are two important differences to the "toy example" above.

1. The data format differs slightly, with the classification being stored in the separate `.labels` rather than having two different files for normal and anomalous data.

[3] http://www.cs.unm.edu/~immsec/systemcalls.htm

2. More importantly, the sequences stored in the files are no longer of a fixed length. For training, this means that you will need to pre-process each sequence to a set of fixed-length chunks (for instance, you could use all substrings of a fixed length, or all non-overlapping substrings of a fixed length). For classification, you also need to split the sequences into chunks, compute the number of matching patterns for each chunk separately, and merge these counts together to a composite anomaly score (for instance, you could average the individual counts).
3. In this exercise, we have provided you an implementation of the negative selection algorithm. In your own words, explain what the biggest challenges would be, were you to implement the algorithm for yourself for the given task.

Choose the parameters n and r for the negative selection algorithm yourself. You can use the parameters from the language example as a starting point.

Important notes

If running negative selection on one set of parameters during training takes too long, then you might run into time trouble. Running negative selection on one set of parameters and plotting the resulting ROC takes under a minute on my system. Get in touch if it takes you much longer. Make sure to think about how to use the data you have effectively (think e.g. splitting into training and test sets).

If you encounter any problems or if you have any questions, please let us know and we will get back to you as soon as possible. From the above, it should be obvious what you have to do – if it is taking you unreasonably long, do reach out in time!

Report

Write a brief report on this assignment (~ 3 pages in pdf). Your report should mostly address the second part of the assignment (intrusion detection), but please include the answers to the questions of the language classification example in an appendix (i.e. your response to "Your task" on page 2).

Don't go out of your way to stick to the page limit, but there's no need to write anything very detailed as long as you cover the points mentioned below.

Assessment criteria

- Design: to what extent does the setup of your experiment(s) allow you to fulfil the task posed in the assignment, or did you miss relevant factors?
- Methodology/implementation: there should be no major bugs/artefacts in your implementation, and the details of your implementation (e.g. choice of n and r, preprocessing) should be clear and justified from the written report. You should still refer to the code, but we should not need it to understand what you did.
- Analysis: how well is the report supported by visualizations and/or quantitative analyses, and are they appropriately chosen?
- Interpretation: Did you correctly interpret your results? Did you clearly answer the question, and are all your claims backed up by evidence?
- Reporting: does your report meet basic standards of a research report? It should be comprehensible and follow a logical structure with an introduction containing the problem statement, methods, results, discussion and conclusion (although you can keep introduction and discussion very brief for this assignment).