Natural Computing
Assignment 2

Christoph Schmidl
s4226887
`c.schmidl@student.ru.nl`

Koen Vijverberg
s4132858
`koen.vijverberg@student.ru.nl`

Alex Kolmus
s4125304
`alex.kolmus@student.ru.nl`

February 26, 2017

Anomaly Detection using Negative Selection Algorithms

1. **Using the Negative Selection Algorithm**

    1. Compute the area under the receiver operating characteristic cuve (AUC) to quantify how well the negative selection algorithm with parameters $n = 10, r = 4$ discriminates individual English string from Tagalog strings by using the files **english.train** for training and **english.test** as well as tagalog.test for testing.
    Solution:

    The code regarding this assignment can be found at our Git Repository:

    - `https://github.com/ChristophSchmidl/natural-computing/blob/master/assignment-2`

    For $n = 10, r = 4$ the $AUC = 0.79$

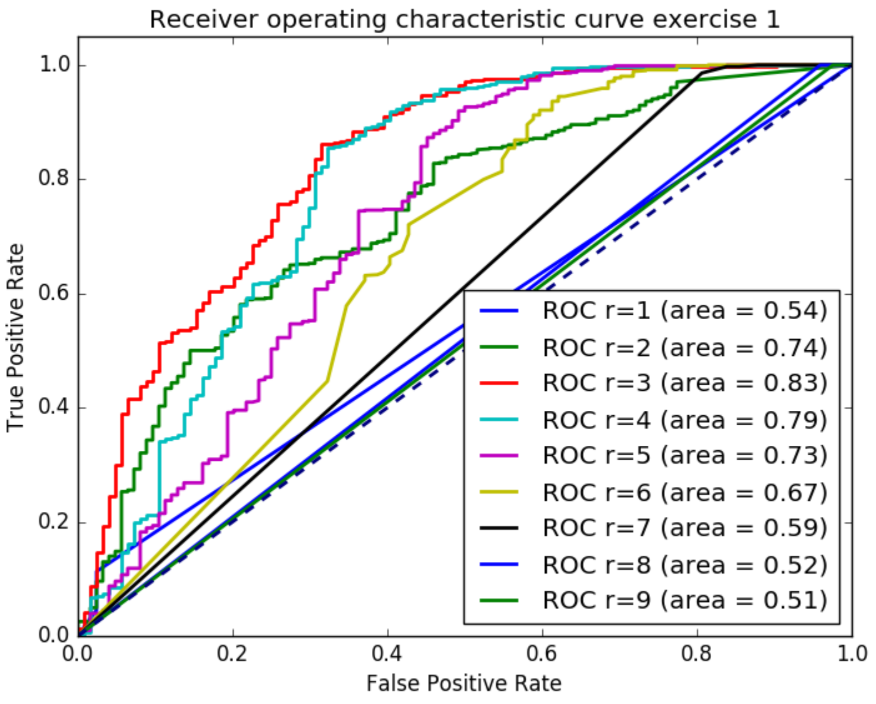

Figure 1: The ROC curve plot for different values of r, the subsequent character matching parameter

2. How does the AUC change when you modify the paramter r? Specifically, what behavior 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?
   Solution:

   See figure 1, the AUC is in both edge cases ($r = 1$ and $r = 9$) close to 0.5, a.k.a. not better than random guessing. Best value is for $r = 3$ with an $AUC = 0.83$.

3. The folder **lang** contains strings from 4 other languages. Determine which of these languages can be best discriminated from English using the negative selection algorithm, and for which of the languages this is most difficult. Can you explain your findings?
   Solution:

   Xhosa is easiest discriminated (best AUC score), worst is middle-english, which seems reasonable because it is probably similar to the normal english language.

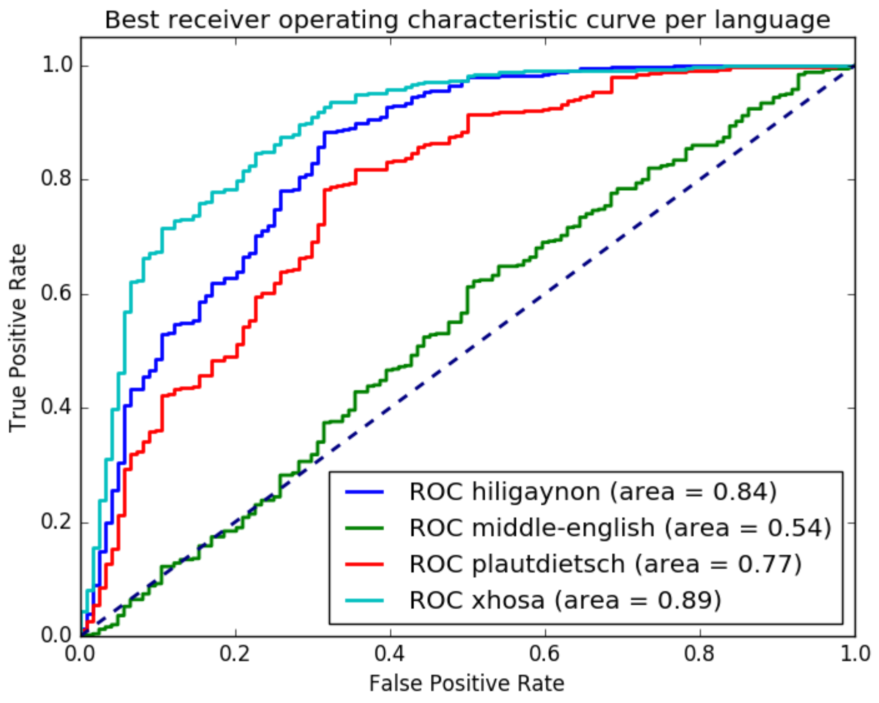

Figure 2: Best ROC curves per language

2. **Intrusion Detection for Unix Processes**

Use negative selection to detect the anomalous sequences in the system calls datasets! Perform an AUC analysis to evaluate the quality of your classification.
There are two importan differences to the "toy example".

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.
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).

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.
Solution:

For n we chose 10 and took r variable. To get our ensure that all input had the correct length, all input lines were split into non-overlapping chunks of size 10. If a chunk had a size smaller then 10, it was appended with minuses. During the classification each chunk was scored, the score for each original input was the average over its chunks. Below the ROC curves of all test data can be seen.

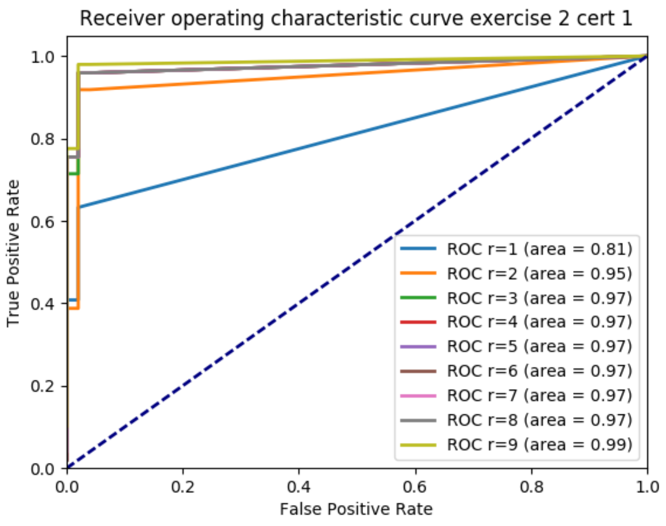

Figure 3: The ROC curve plot for different values of r for test set cert 1.

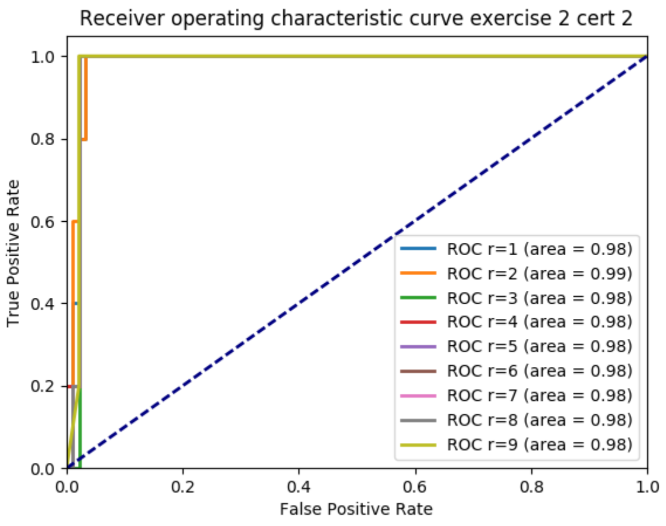

Figure 4: The ROC curve plot for different values of r for test set cert 2.

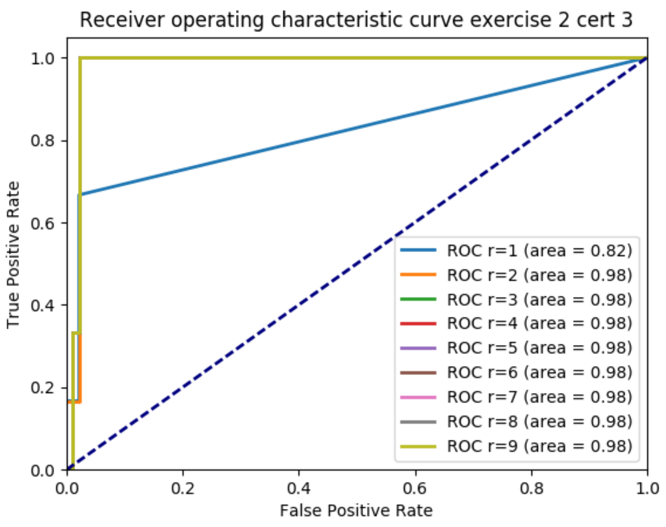

Figure 5: The ROC curve plot for different values of r for test set cert 3.

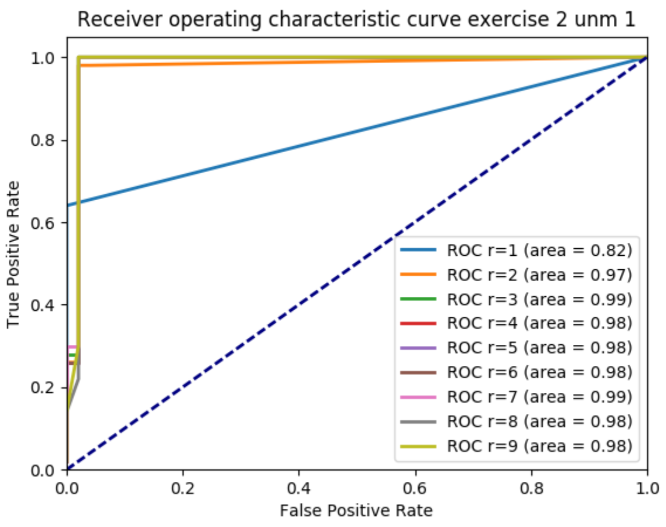

Figure 6: The ROC curve plot for different values of r for test set unm 1.

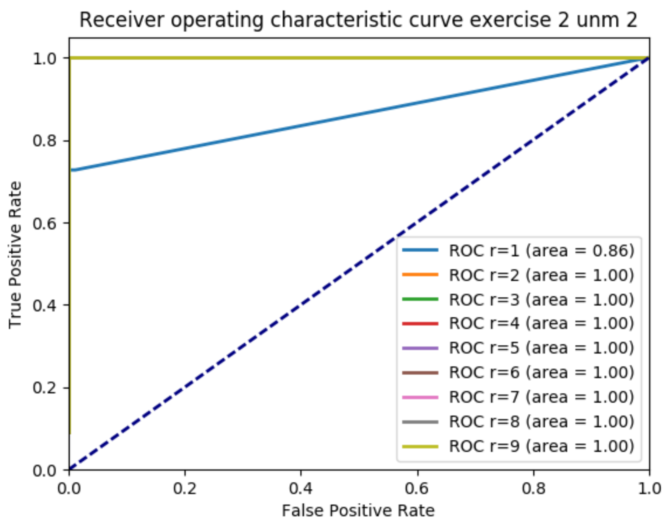

Figure 7: The ROC curve plot for different values of r for test set unm 2.

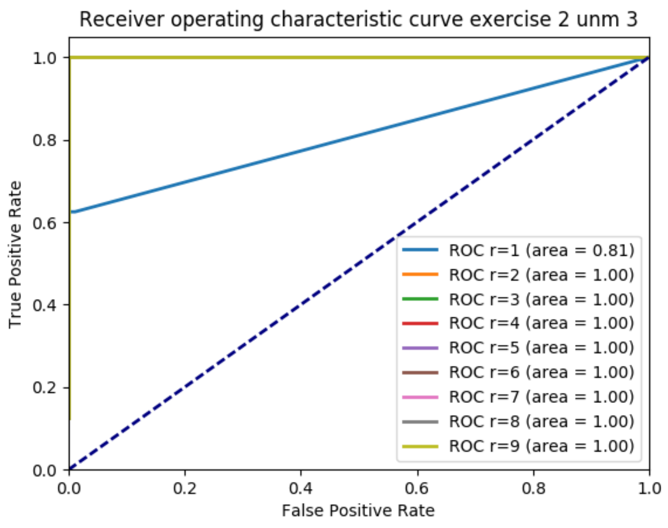

Figure 8: The ROC curve plot for different values of r for test set unm 3.

Although odd it appears that for $n = 10$ and $r > 2$ the classification is close to perfect. The code regarding these exercises can be found at our Git Repository:

`https://github.com/ChristophSchmidl/natural-computing/blob/master/assignment-2/`