Natural Computing
Assignment 2

Tim van Dijk - `S4477073`
Tristan de Boer - `S1007313`

February 2018

1 Using the Negative Selection Algorithm

1.1 Using the code given in Appendix A, with parameters $n = 10$ and $r = 4$, we obtain $AUC = 0.792$ for Figure 1.

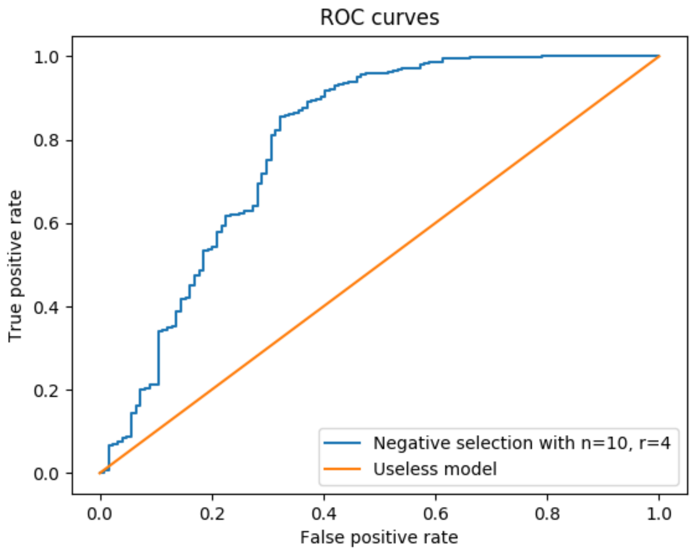


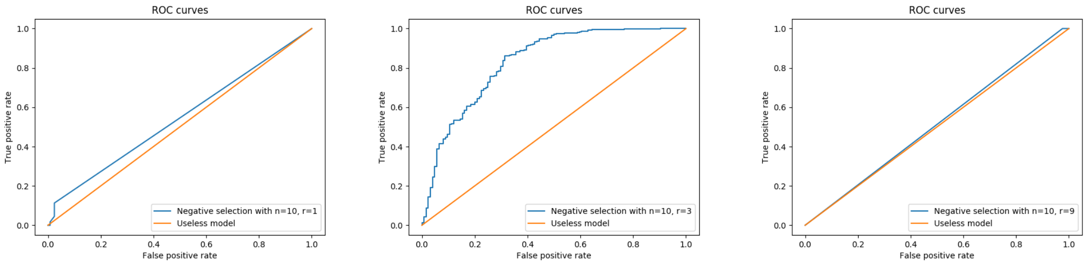
Figure 1: ROC plot with $n = 10$ and $r = 4$

1.2 Using $r = \{1, 2, ..., 9\}$, we obtain the following area under the ROC:

- For n=10 and r=1, the area under the ROC curve is: 0.544
- For n=10 and r=2, the area under the ROC curve is: 0.74
- For n=10 and r=3, the area under the ROC curve is: 0.831
- For n=10 and r=4, the area under the ROC curve is: 0.792
- For n=10 and r=5, the area under the ROC curve is: 0.728
- For n=10 and r=6, the area under the ROC curve is: 0.668
- For n=10 and r=7, the area under the ROC curve is: 0.591
- For n=10 and r=8, the area under the ROC curve is: 0.52

- For n=10 and r=9, the area under the ROC curve is: 0.512

For $r = \{1, 3, 9\}$, we added the plots of the ROC in Figure 2.



(a) $r = 1$ (b) $r = 3$ (c) $r = 9$

Figure 2: ROC plots with $r = 10$ and $r = \{1, 3, 9\}$

The models where $r = 1$ and $r = 9$ both perform very poorly, but for different reasons. The former performs poorly because we are looking to match continguous strings of length 1, which often is the case (by chance) when we try to classify inputs that merely use the same alphabet as the training set. The latter performs poorly because we are overfitting by trying to match very long continguous strings.

$r = 3$ yields the best results as the area under the ROC curve is largest, namely 0.831.

1.3 Using the code given in Appendix B, the following results were obtained.

- For hiligaynon, MAX(AUC) = 0.84
- For middle-english, MAX(AUC) = 0.542
- For plautdietsch, MAX(AUC) = 0.775
- For xhosa, MAX(AUC) = 0.889

The model performs best on xhosa, and worst on middle-english. If we inspect the corpus of xhosa, we encounter strings such as:

```
bini_babuy
ngadyobhek
qinela_uku
halo_nalil
esi_sifo_a
lo_uyesu_k
```

The combination of letters is very different from those in English. There aren't many words in english that contain substrings such as "dyo" or "uye". Hence, the words are seen as non-self.

The middle-english corpus is much more similar to English:

```
m_what_sek
hal_be_if_
we_he_wold
_haue_spok
```

We see letter combinations that we encounter often in English and some words even are part of the English language. Therefore, it is much more difficult to distinguish self from non-self.

2 Intrusion Detection for Unix Processes

We use negative selection to detect the anomalous sequences in the system calls datasets! We 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.

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 We chose $n = 10$ and took $r = \{1, 2, ..., 9\}$. All input was pre-processed into a set of fixed-length non-overlapping chunks of size 10. If a chunk was smaller than size 10, it was appended with minuses. During the classification we scored each chunk by taking the average over its chunks. The acquired ROC curves can be seen below.

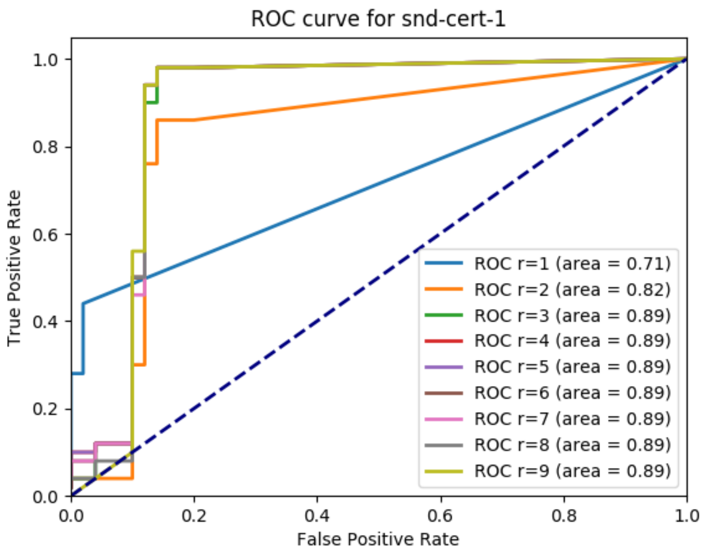


Figure 3: ROC curve for snd-cert-1

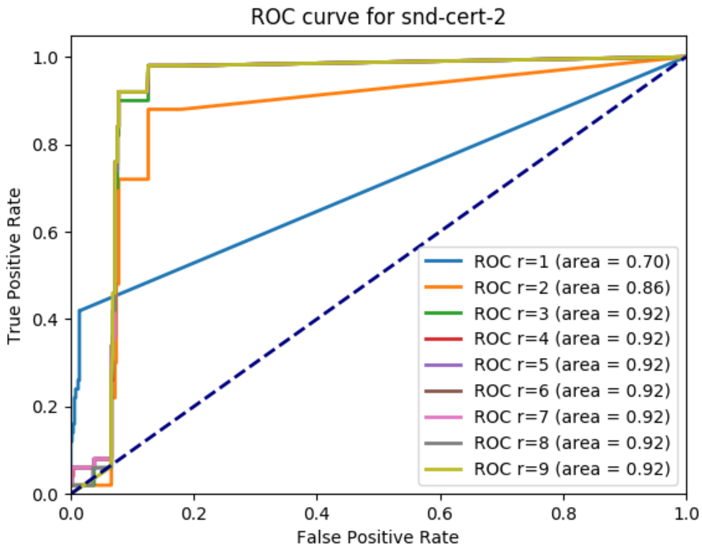


Figure 4: ROC curve for snd-cert-2

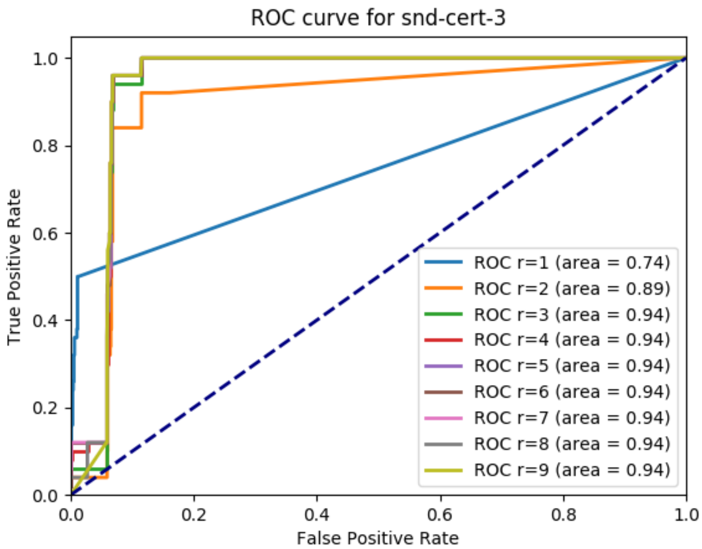


Figure 5: ROC curve for snd-cert-2

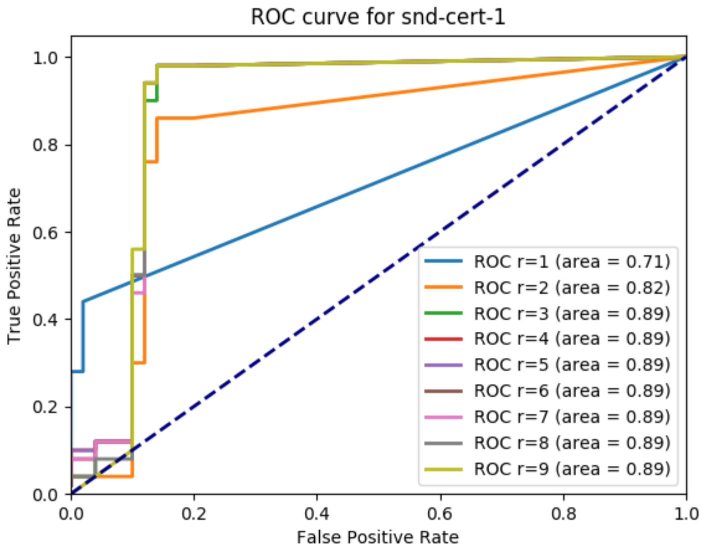


Figure 6: ROC curve for snd-unm-1

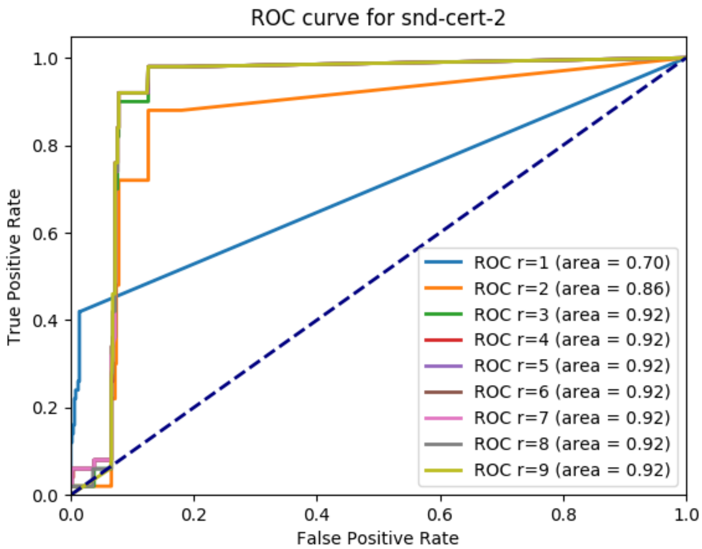


Figure 7: ROC curve for snd-unm-2

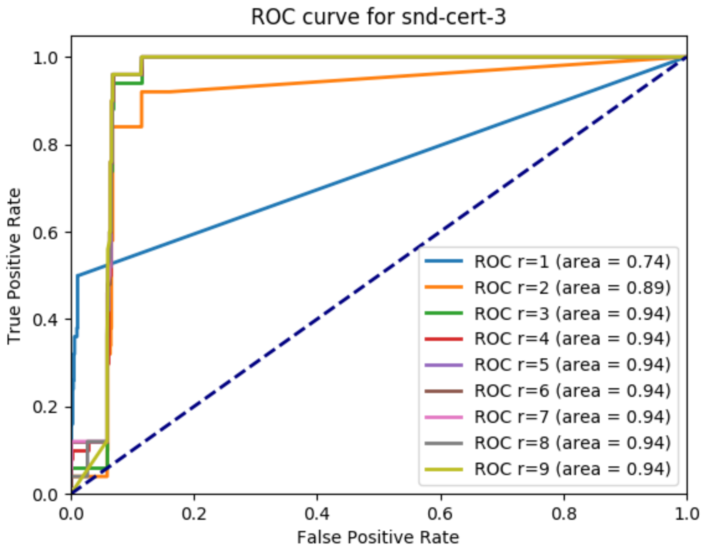


Figure 8: ROC curve for snd-unm-3

When using $r > 2$, the accuracy of the classification does not improve much. All curves contain some error around $0 < FPR < 0.1$. We believe it is possible to get rid of this error, but we do not know how.

As the code requires three files to run (that are all to big to include in this report), we made the files available on here: https://github.com/tristandb/NaturalComputing/tree/master/A2/E2.

A AUC.py

```
import matplotlib.pyplot as plt
import os

for r in range(1, 10):
        os.system("java -jar negsel2.jar -self english.train -n 10 -r
            {} -c -l < english.test > english.results.{}".format(r, r))
        os.system("java -jar negsel2.jar -self english.train -n 10 -r
            {} -c -l < tagalog.test > tagalog.results.{}".format(r, r))

        with open("english.results.{}".format(r)) as f:
                e_data = f.read()

        with open("tagalog.results.{}".format(r)) as f:
                t_data = f.read()

        e_scores = [int(2**float(s)) for s in e_data.split(' \n') if s]
        t_scores = [int(2**float(s)) for s in t_data.split(' \n') if s]

        tprs, fprs = [0], [0]
        for threshold in sorted(list(set(e_scores + t_scores)))[::-1]:
                tprs += [len([score for score in t_scores if score >=
                    threshold]) / len(t_scores)]
                fprs += [len([score for score in e_scores if score >=
                    threshold]) / len(e_scores)]

        area = sum([(fprs[i]-fprs[i-1]) * (tprs[i-1]+tprs[i])/2 for i
            in range(1, len(tprs))])
        print("For n=10 and r={}, the area under the ROC curve is: {}".
            format(r, round(area, 3)))

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.set_title("ROC curves")
        ax.set_xlabel('False positive rate')
        ax.set_ylabel('True positive rate')

        roc = plt.plot(fprs, tprs, label="Negative selection with n=10,
             r={}".format(r))
        avg = plt.plot([0, 1], [0, 1], label="Useless model")

        ax.legend(loc=4)
        plt.savefig("ROC_{}.png".format(r))
        #plt.show()
        plt.close(fig)
```

B Compare languages

```
import matplotlib.pyplot as plt
import os

for lang in ['hiligaynon', 'middle-english', 'plautdietsch', 'xhosa']:
        for r in range(1, 10):
                os.system("java -jar negsel2.jar -self english.train -n
                     10 -r {} -c -l < english.test > english.results.{}
```

```
    ".format(r, r))
os.system("java -jar negsel2.jar -self english.train -n 10 -r {} -c -l < lang/{}.txt > {}.results.{}".format(r, lang, lang, r))

with open("english.results.{}".format(r)) as f:
        e_data = f.read()

with open("{}.results.{}".format(lang, r)) as f:
        l_data = f.read()

e_scores = [int(2**float(s)) for s in e_data.split(' \n') if s]
l_scores = [int(2**float(s)) for s in l_data.split(' \n') if s]

tprs, fprs = [0], [0]
for threshold in sorted(list(set(e_scores + l_scores)))[::-1]:
        tprs += [len([score for score in l_scores if score >= threshold]) / len(l_scores)]
        fprs += [len([score for score in e_scores if score >= threshold]) / len(e_scores)]

area = sum([(fprs[i]-fprs[i-1]) * (tprs[i-1]+tprs[i])/2 for i in range(1, len(tprs))])
print("For lang={}, n=10 and r={}, the area under the ROC curve is: {}".format(lang, r, round(area, 3)))

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title("ROC curves")
ax.set_xlabel('False positive rate')
ax.set_ylabel('True positive rate')

roc = plt.plot(fprs, tprs, label="Negative selection with n=10, r={}".format(r))
avg = plt.plot([0, 1], [0, 1], label="Useless model")

ax.legend(loc=4)
plt.savefig("ROC_{}_{}.png".format(lang, r))
#plt.show()
plt.close(fig)
```