Steven Smits
s4232763

Tjalling Haije
s1011759

Tido Bergmans
s4373561

Kai Standvoss
s4751744

Four Horsemen — Assignment II
Natural Computing

Anomaly Detection using Negative Selection Algorithms

1. Using the Negative Selection Algorithm

1. Compute the area under the receiver operating characteristic curve (AUC) to quantify how well the negative selection algorithm with parameters n = 10,r = 4 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.

   Answer: The negative selection algorithm performs the binary classification at an AUC of 0.79. See Figure 1 for the ROC. The code following the figure was used.

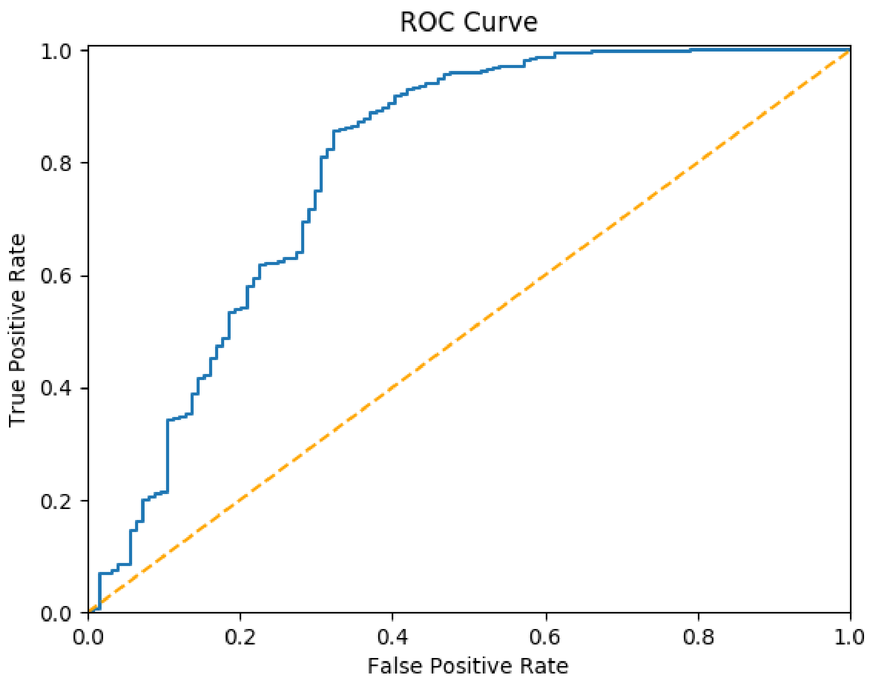

Figure 1: ROC curve for classifying English as normal and Tagalog anomalous using negative selection at n=10 and r=4. The AUC was 0.79.

```python
import numpy as np
from matplotlib import pyplot as plt
from subprocess import run, PIPE
import os

## REQUIRES PYTHON3

def negsel(test_file, r):
        '''
        Performs negative selection algorithm by calling negsel2.jar
        Parameters:
        ---------
        test_file: str
                filename of file containing test data
        r: int
                max len of contingouos substring matches
        Returns:
        ---------
        array_like
                array containing log number of pattern matches
        '''
        args = ['java', '-jar', 'negsel2.jar', '-self',
        ↪ 'english.train', '-n', '10', '-r', str(r), '-c', '-l']
        with open(test_file, 'r') as f:
                lines = f.read()
                p = run(args, stdout=PIPE, input=lines,
                ↪ encoding='ascii')
                res = np.fromstring(p.stdout.replace('\n',''),
                ↪ dtype=float, sep=' ')
        return res


def cal_roc(scores_true, scores_false):
        '''
        Calculate sensitivity and specificity for roc analysis
        Parameters:
        ---------
        scores_true: array_like
                negsel scores for correct class
        scores_false: array_like
                negsel scores for anomalous class
        Returns:
        ---------
        array_like
                sensitivity
        array_like
                specificity
        '''
        scores = np.concatenate([scores_true,scores_false])
        uniques = np.unique(scores)
        roc = np.zeros((len(uniques),2))
        for i,u in enumerate(uniques):
                higher = np.where(scores >= u)[0]
                sens = np.sum(higher >= len(scores_true)) /
                 ↪  len(scores_false)
                lower = np.where(scores < u)[0]
                spec = np.sum(lower < len(scores_true)) /
                 ↪  len(scores_true)
                roc[i] = [sens,spec]
        return roc[:,0], roc[:,1]

def calc_auc(sensitivity, specificity):
        '''
        Calculate area under the curve
        using trapezoidal approximation for the integral
        Parameters:
        ---------
        sensitivity: array_like
                sensitivity scores
        specificity: array_like
                specificity scores
        Returns:
        ---------
        float
                AUC
        '''
        #return negative value since arrays are
        #sorted in descending order
        return -np.trapz(sensitivity,1-specificity)

#perform negsel on english and tagalog test set for english training
 ↪  data
r = 4
res_eng = negsel('english.test', r)
res_tag = negsel('tagalog.test', r)

#Perform ROC analysis for r = 4
```

```
sens, spec = cal_roc(res_eng,res_tag)
auc = calc_auc(sens,spec)
print('AUC for r = 4 is {}'.format(auc))

plt.figure()
plt.plot(1-spec,sens)
plt.plot(np.arange(0,1.1,0.1),np.arange(0,1.1,0.1), color='orange',
 ↪  linestyle='--')
plt.xlim([0,1])
plt.ylim([0,1.01])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```

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?

   Answer: The ROC curves and corresponding AUC's were calculated for $r = 1, .., 9$ (Figure 2 and Figure 3 respectively). At r=1, the negative selection algorithm performs at chance level with an AUC $\tilde{0}.50$.At r=9 the AUC is $\tilde{0}$, which means there is almost no chance that a normal (English) test example ranks above a anomalous (Tagalog) test example (or in negative selection terms: the opposite). At r=9 the algorithm matches substrings of more than 9 (so 10) with the repertoire that contains patterns of length 10. So testing strings must exactly match training strings to be detected as such, which the algorithm can't find. Thus, the negative selection algorithm classifies all testing examples as anomalous. The figures suggest that r=3 leads to the best discrimination. The following code was used:

```
#Plot ROC curves
plt.figure()
plt.plot(np.arange(0,1.1,0.1),np.arange(0,1.1,0.1), color='orange',
 ↪  linestyle='--')
plt.xlim([0,1])
plt.ylim([0,1.01])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves given r')

aucs = []
for r in range(1,10):
        res_eng = negsel('english.test', r)
        res_tag = negsel('tagalog.test', r)

        sens, spec = cal_roc(res_eng,res_tag)
        plt.plot(1-spec,sens, label='r = {}'.format(r))
        aucs.append(calc_auc(sens,spec))

plt.legend()

#Plot AUC against r
plt.figure()
plt.plot(np.arange(1,10),aucs)
plt.xlabel('r')
plt.ylabel('AUC')
plt.title('AUC for differen values of r')
```

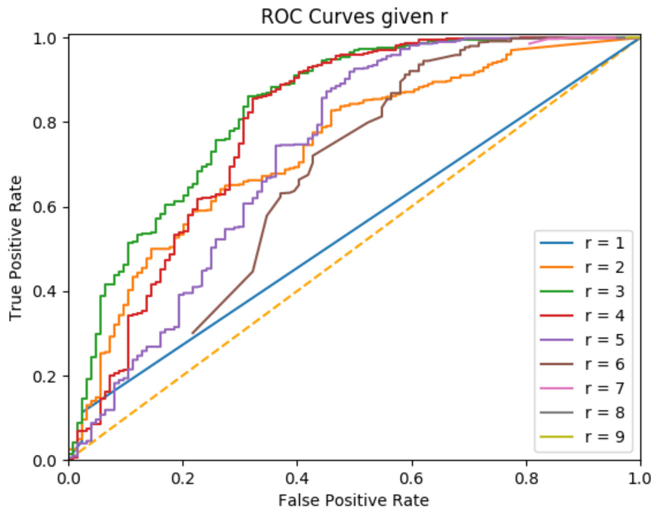

Figure 2: ROC curves for classifying English as normal and Tagalog anomalous using negative selection at n=10 and varying r.

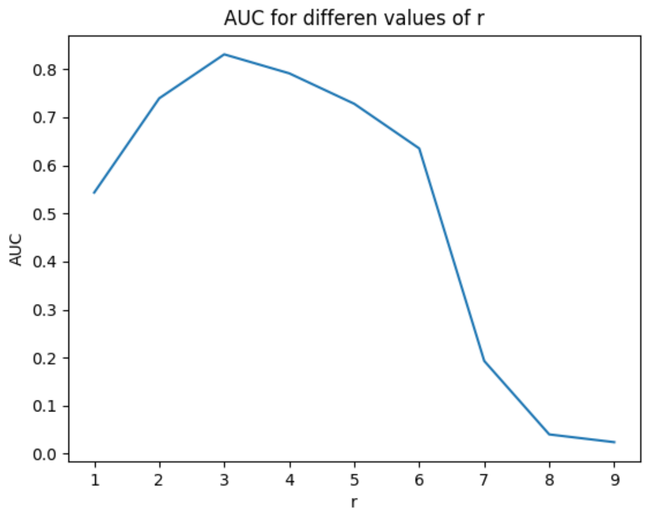

Figure 3: AUC's for classifying English as normal and Tagalog anomalous using negative selection at n=10 and varying r.

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?

   Answer: At most r values (but specifically at r=3 that was found to be maximal discriminating), the Xhosa language is best discriminated from the English language using negative selection (see Figure 4, AUC $\approx$ 0.80). The middle English language was most difficult to distinguish from English (AUC $\approx$ 0.5). That th English language is hardest to discriminate comes as no surprise, since the spelling of modern English is derived from middle English. The Xhosa language is actually written in the Latin alphabet (not in the test text file though) and is a language from Zimbabwe, which means it has not been influenced by the English language that much. Thus English and Xhosa are "maximally" different, whereas English and middle English are minimally different. The following code was used for varying r values:

```
r = 3 # But varying
res_eng = negsel('english.test', r)
print('{:^20}|{:^8}'.format('Language', 'AUC'))
print('--------------------|--------')
for lang in os.listdir('lang/'):
        res_lang = negsel('lang/' + lang, r)
        sens, spec = cal_roc(res_eng,res_lang)
        auc = calc_auc(sens,spec)
        print('{:^20}|{:^8.4}'.format(lang[:-4],auc))
```

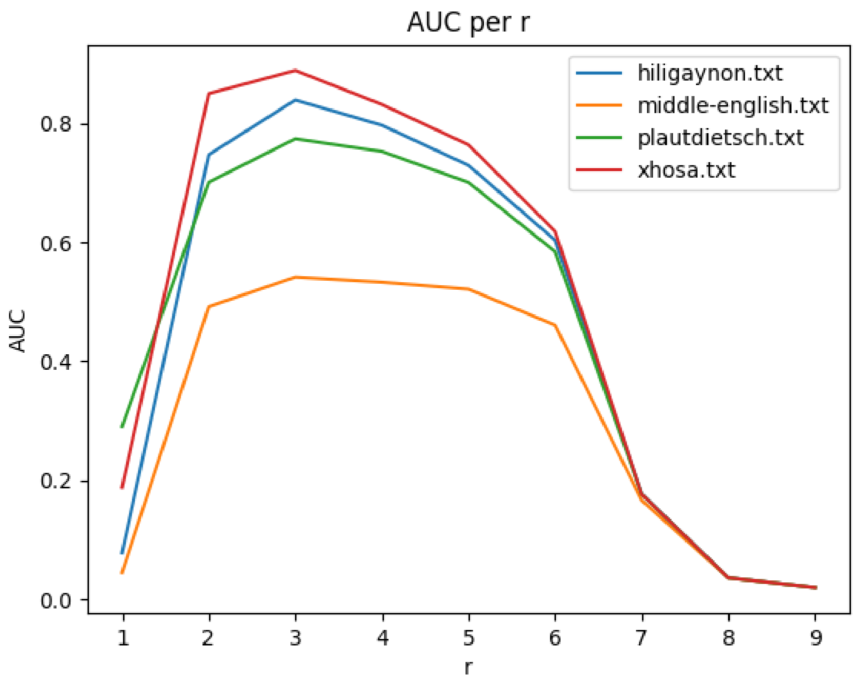

Figure 4: AUC's for classifying English as normal and multiple languages (separately) as anomalous using negative selection at n=10 and varying r.

2. Intrusion Detection for Unix Processes

1. 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 important differences to the "toy example" above

   (a) 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.

   (b) 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.

   Answer: The negative selection algorithm is able to discriminate anomalous from normal sequences in system calls with a performance roughly in at $\tilde{0}.90$ (see Table 1 for specific file-type testing). The ROC curves are plotted for both types of system calls their first testing data set (see Figure 5 and Figure 6). These are representative performance on the other testing files as well. The figures depict that the negative selection algorithm can maintain to be very sensitive to anomalous calls while not erroneously classifying normal calls as anomalous (Type I error).

File #	**snd-crt**	**snd-unm**
1	0.92	0.90
2	0.92	0.89
3	0.93	0.92

Table 1: AUROC's for the negative selection algorithm classifying anomalous sequences in the system calls datasets (snd-crt and snd-unm) at n=10 ad r=4.

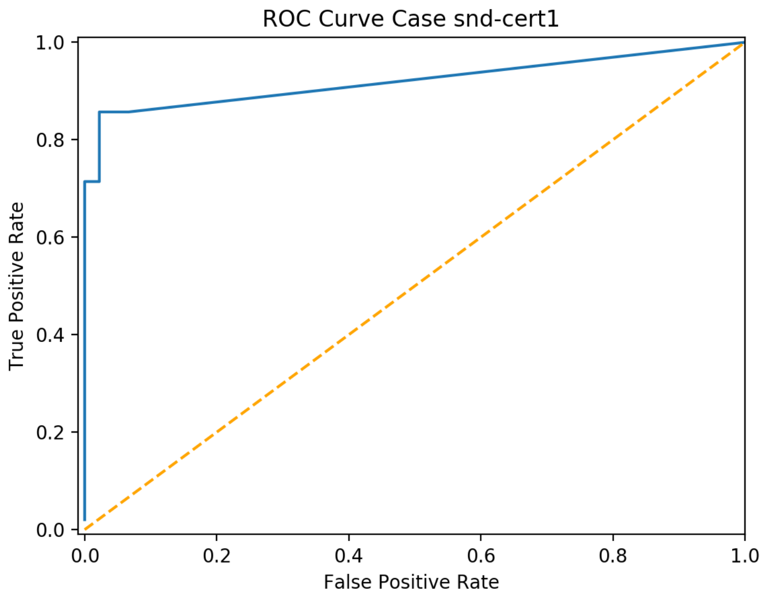

Figure 5: ROC curve for case snd-cert1

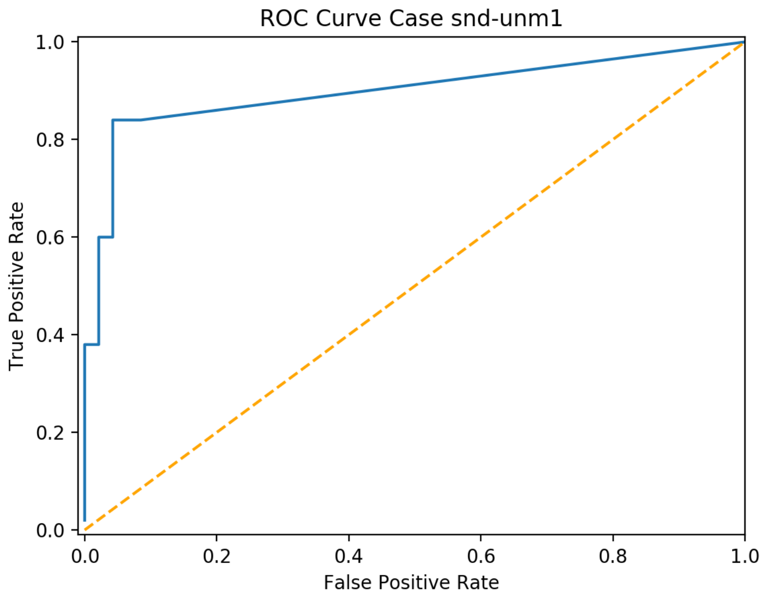

Figure 6: ROC curve for case snd-unm1

```python
import numpy as np
from matplotlib import pyplot as plt
from subprocess import run, PIPE
import os
from pathlib import Path
import negsel
import pdb

## REQUIRES PYTHON3
## Takes very long to run!

def chunk(data, length, overlap=False):
        '''
        Function to split data into chunks
        :param (array_like)         data
        :param (int) length         length of chunks
        :param (bool) overlap          whether chunks should overlap
 ↪  or not
        :return (array_like) chunked data
        '''
        chunk_data = []
        step = 1 if overlap else length
        for i in range(0,len(data)-length,step):
                chunk = data[i:i+length]
                chunk_data.append(chunk)
        return chunk_data


## If training data is not yet chunked create new file with chunked
 ↪  data
CHUNK_LEN = 10

#SUBFOLDER = 'snd-cert'
SUBFOLDER = 'snd-unm'
TRAIN_PATH = Path('syscalls/' + SUBFOLDER)
DATA = SUBFOLDER + '.train'
CHUNK_PATH = TRAIN_PATH / 'chunks.train'
if not CHUNK_PATH.exists():
        with open(TRAIN_PATH / DATA, 'r') as f:
                syscalls = f.read()
                syscalls = syscalls.replace('\n','')

        chunk_data = chunk(syscalls,CHUNK_LEN)
        with open(CHUNK_PATH, 'w') as f:
                f.writelines("%s\n" % c for c in chunk_data)

## Apply to test data
r = 4
TEST_PATH = Path('syscalls/' + SUBFOLDER)
TEST_CASES = 1
for i in range(1,TEST_CASES+1):
        #read data
        case_data = SUBFOLDER + '.' + str(i) + '.test'
        with open(TEST_PATH / case_data, 'r') as f:
                syscalls = np.array(f.readlines())

        #read labels
        case_labels = SUBFOLDER + '.' + str(i) + '.labels'
        with open(TEST_PATH / case_labels, 'r') as f:
                labels = f.readlines()
        labels = np.array(labels).astype(bool)


        def call_negsel(data):
                '''
                Helper function to call negative selection
                :param data
                :return scores
                '''
                test_data = []
                lens = []
                #chunk test data and save number of chunks per line
                for line in data:
                        chunks = chunk(line[:-1],CHUNK_LEN)
                        lens.append(len(chunks))
                        test_data.extend(chunks)

                #apply negative selection
                res = negsel.negsel(CHUNK_PATH, test_data,
                 ↪ CHUNK_LEN, r)

                #take mean score per line in test_data
                lens = np.array(lens)
                scores = []
                start = 0
                for l in lens[lens>0]:
                        scores.append(np.mean(res[start:start+l]))
                        start += l

                return scores

        ## Apply negative selection to positive and negative samples
         ↪  separately
        pos_scores = call_negsel(syscalls[labels])
        neg_scores = call_negsel(syscalls[~labels])

        sens, spec = negsel.cal_roc(neg_scores, pos_scores)
        auc = negsel.calc_auc(sens,spec)

        ## Plot
        print('AUC for r = 4 is {}'.format(auc))

        plt.figure()
        plt.plot(1-spec,sens)
        plt.plot(np.arange(0,1.1,0.1),np.arange(0,1.1,0.1),
         ↪  color='orange', linestyle='--')
        plt.xlim([-.01,1])
        plt.ylim([-.01,1.01])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC Curve Case ' + SUBFOLDER + str(i))


plt.show()
```