Assignment 3 - Exercise 2

In [1]:
```
import subprocess

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score, roc_curve
```

In [2]:
```
def load_data(filename:str):
    """
    Helper function that loads data from storage given
    a filename that contains a full path to the file.
    Assumes that each datapoints is stored as individual line
    in the file.
    """
    data_list = []
    with open(filename, 'r') as f:
        for line in f:
            data_list.append(line[:-1])
    return data_list
```

Set path variables below that denote which data to use

In [3]:
```
# Directory that contains everything related to this exercise
syscalls_dir = 'syscalls/'

# Specify the name of the dataset ('snd-cert' or 'snd-unm')
dataset_name = 'snd-cert'

# Specify which test dataset to use (1, 2, or 3 for each of the
# above datsets)
testdata_number = 1

# All data files have the same sub-path so we can reuse it
path_to_data_files = syscalls_dir+dataset_name+'/'+dataset_name
```

Prepare Datasets

In [4]:
```
# Load train data
train_name = path_to_data_files+'.train'
train_data = load_data(train_name)
```

In [5]:
```
# Load test data
test_name = path_to_data_files+f'.{testdata_number}.test'
test_data = load_data(test_name)
test_data_len = len(test_data)

# Load test labels
test_labels = path_to_data_files+f'.{testdata_number}.labels'
test_labels = load_data(test_labels)
```

Get the minimum sequence length of both train and test data. This is needed for computing the chunks later on.

In [6]:
```
min_seq_len = min(
    min([len(s) for s in train_data]),
    min([len(s) for s in test_data])
)
print('Shortest sequence in both train and test set:',
      min_seq_len)
```

```
Shortest sequence in both train and test set: 7
```

Chunk Datasets

In [7]:
```
def chunk_sequence(sequence:str, chunk_size:int):
    """
    Chunk a sequence into subsequences of length <chunk_size>.
    If the last subsequence is shorter than <chunk_size> it is
    dropped.
    Note: we assume that the sequence is at least of length
    <chunk_size> because we compute the minimum sequence length
    above. If it is shorter, this function returns an empty list.
    """
    # Special case for short sequences
    if len(sequence)==chunk_size:
        return [sequence]

    # Compute chunks from sequence
    chunks = []
    for i in range(chunk_size, len(sequence), chunk_size):
        chunks.append(sequence[i-chunk_size:i])

    return chunks
```

Below, we chunk each sequence in the training data into chunks of specified length `min_seq_len`. We use list comprehension to do this is a fast way, an simultanously flatten the list. Thus, `train_data` will be a non-nested list of sequences of length `min_seq_len`. Additionally, we convert to a `set` since we do not need duplicate sequences.

In [8]:
```
train_data = list(set(
    [seqq
     for seq in train_data
     for seqq in chunk_sequence(seq, min_seq_len)]))
```

We do the same below for the test data. Note however, that we cannot simply chunk the data and flatten the list like above, because we need to keep track of the original sequence a chunk belongs to in order to later on compute the average score over all chunks of a sequence.

To achieve this, we first chunk all sequences in `test_data` without flattening the list.

Then, we flatten the list, but record the index of the original sequence together with the actual chunk string as a tuple.

In [9]:
```
# Chunk all sequences without flattening
test_data = [chunk_sequence(seq, min_seq_len)
             for seq in test_data]
# Flatten the list but record chunk together with index of sequen
ce
# it belongs to
test_data = [(chunk,idx)
             for idx,sublist in enumerate(test_data)
             for chunk in sublist]
```

Save Data to Disk

We need to save the datasets to disk such that the call to Java can pick them up

In [10]:
```
# Write chunked train data to file
with open(train_name+'.chunked', 'w') as f:
    for line in train_data:
        f.write(line)
        f.write('\n')
```

In [11]:
```
# Write chunked test data to file
with open(test_name+'.chunked', 'w') as f:
    for chunk, idx in test_data:
        f.write(chunk)
        f.write('\n')
```

Run Algorithm

First we define the function to run the algorithm given a train and test set name, and the sequence length

In [12]:
```
def get_scores(train_name, test_name, seq_length, r=4):
    """
    Run the Negative Selection algorithm implemented in Java.
    This issues a system call to a subprocess with the
    arguments needed to run the Java program.

    PARAMS
    ======
    train_name: The file name (full path) to the training set
    test_name: The file name (full path) to the test set
    seq_length: The length of the sequences in the sets
    r: Parameter r of the Negative Selection algorithm

    RETURNS
    =======
    The score for each of the datapoints in the testset
    """
    # Define the command to run the algorithm with Java
    run_command = \
        f"java -jar negsel2.jar -self {train_name} " \
        f"-n {seq_length} -r {r} -c -l < {test_name}"
    # Issue call to subprocess to run the command
    results = subprocess.getoutput(run_command)
    # Convert the results to numpy array of floats
    return np.array([float(r) for r in results.split('\n')])
```

In [13]:
```
scores = get_scores(
    train_name+'.chunked',
    test_name+'.chunked',
    seq_length=min_seq_len)
```

Right now, the scores are a one-dimensional list, containing a score for each chunk. We now want to map these chunk-scores back to the sequence they belong to, and then compute the average score per sequence.

To do this, we first create a list `unnested_scores` that contains an empty list for every element in the original test data. Then, we populate this list by appending the score from `scores` at the correct index. We obtain the correct index from the `test_data` that we populated with tuples in cell 9. Lastly, we compute the average for each sequence simply by taking the mean of each of the lists in `unnested_scores`.

In [14]:
```
# Get average scores for sequences from chunks
unnested_scores = [list() for _ in range(test_data_len)]

for i,score in enumerate(scores):
    unnested_scores[test_data[i][1]].append(score)

avg_scores = [np.mean(sublist) for sublist in unnested_scores]
```

Finally, we compute the ROC score for this fit.

In [15]:
```
# Compute ROC score
roc_auc_score(test_labels, avg_scores)
```

Out[15]:
```
0.9779999999999999
```