0.1 Test cases

Table 1: UNIT-01

Item	**Description**
Name	Command line interface (CLI) functionality
Test identifier	UNIT-01
Person responsible	Henrik Knutsen
Feature(s) to be tested	That all possible commands are working correctly when using the CLI. That the program runs without input
Pre-conditions	Code for input handling for all possible commands. Code for error handling for invalid inputs.
Execution steps	1. Run the program for every type of argument 2. Run the program without arguments
Expected results	1. The specified variable is set to the specified value 2. All the values are loaded from the config file

Table 2: UNIT-02

Item	**Description**
Name	P3P parser
Test identifier	UNIT-02
Person responsible	Henrik Knutsen
Feature(s) to be tested	That the P3P parser correctly parses all the required fields and their values.
Pre-conditions	The P3P parser must be implemented. Need to have P3P policies with a wide range of cases.
Execution steps	1. Run a P3P xml in the P3P parser and print the parsed fields and their values to console 2. Manually compare the printed fields and values with the contents of the P3P xml
Expected results	1. The P3P xml is parsed successfully. It's content is printed to console 2. The printed output have the same fields, each having the same value as those in the xml

Table 3: UNIT-03

Item	**Description**
Name	Local database
Test identifier	UNIT-03
Person responsible	Henrik Knutsen
Feature(s) to be tested	Writing to and reading from the local database. That the serialization of the database is working.
Pre-conditions	Code for writing to and reading from the database file. Need to have two different P3P policies.
Execution steps	1. Write policy A to the local database 2. Write policy B to the local database 3. Read and print policy A from the local database 4. Read and print policy A from the local database 5. Compare the written policy A and the read policy A 6. Compare the written policy B and the read policy B

Continued on next page

Table 3 – *Continued from previous page*

Item	**Description**
Expected results	1. Policy A is successfully written to the database file 2. Policy B is successfully written to the database file 3. Policy A is successfully read from the database file and printed 4. Policy B is successfully read from the database file and printed 5. The written policy A and the read policy A are identical. They both have the same fields, with the same values 6. The written policy B and the read policy B are identical. They both have the same fields, with the same values

Table 4: UNIT-04

Item	**Description**
Name	Graphical user interface (GUI) functionality
Test identifier	UNIT-04
Person responsible	Henrik Knutsen
Feature(s) to be tested	That all the elements - buttons, lists etc. - are working as intended.
Pre-conditions	GUI with all the necessary listeners must be implemented. Code for running the program with the GUI.
Execution steps	1. Run the program using the GUI 2. Test every option in the menu bar 3. Test every button in the configuration menu 4. Test every scroll bar 5. Resize the window

Continued on next page

Table 4 – *Continued from previous page*

Item	**Description**
Expected results	1. The program starts and loads the graphical user interface 2. The action connected to the option is executed successfully 3. The action connected to the button/checkbox is executed successfully 4. The scroll bar scrolls through the list, up and down, from end to end, successfully 5. Window can be resized without having elements of the GUI overlapping. The elements and panes scales with the main window

Table 5: UNIT-05

Item	**Description**
Name	Algorithm classification
Test identifier	UNIT-05
Person responsible	Henrik Knutsen, Dimitry Kongevold
Feature(s) to be tested	That the k-nearest neighbor algorithm bases its decision on the k most similar policies
Pre-conditions	Code for reading from the weights file must be implemented. A working k-nearest neighbor algorithm that uses the weights must be implemented. Need one policy to test on, and a set of policies to be used as history.
Execution steps	1. Load a set of policies into the database file 2. Manually calculate and write down the distances between the single policy and each of the policies in the history 3. Run the distance algorithm on a single policy and the history and compare the distances that are returned by the algorithm with the manually calculated distances from step 2 4. Manually find the k policies with the lowest distances 5. Run the reduction algorithm with necessary input to find the k nearest policies and compare the k policies returned by the reduction algorithm with those found in step 4 6. Run the conclusion algorithm and verify the results returned by the algorithm ???

Continued on next page

Table 5 – *Continued from previous page*

Item	**Description**
Expected results	1. The policies are added to the database file 2. The six distances are obtained 3. The algorithm returns the same distances as those found in step 2 4. The k policies are obtained 5. The algorithm returns the same k policies as those found in step 4 6. ??

Table 6: UNIT-06

Item	**Description**
Name	Algorithm learning
Test identifier	UNIT-06
Person responsible	Henrik Knutsen, Neshahavan Karunakaran
Feature(s) to be tested	That the weights file is updated when a new policy is added to history.
Pre-conditions	Code for reading from and writing to the weights file. Code for writing to the database. Algorithms for classification and learning must be implemented.
Execution steps	1. Make a set of policies and load the set into the history 2. Read the weights from the weights file 3. Run the classification and learning algorithms on the policy to be classified and the history, with the weights from step 2 4. Read the weights from the weights file 5. Compare the contents of the weights les obtained in steps 2 and 4

Continued on next page

Table 6 – *Continued from previous page*

Item	**Description**
Expected results	1. The policies are loaded into the history successfully 2. The weights are written down 3. The classification and learning algorithm runs successfully on the policy to be classified and the history 4. The weights are loaded 5. The weights loaded in step 4 are different from the weights written down in step 2

Table 7: UNIT-07

Item	**Description**
Name	Interaction with community databas
Test identifier	UNIT-07
Person responsible	Henrik Knutsen
Feature(s) to be tested	That packets can be sent between the client program and the community database.
Pre-conditions	A running local client. A (virtual) server. Code for sending and receiving packets must be implemented.
Execution steps	1. a
Expected results	1. a