Testing Report

Frankland, Matthew Hafiz, Farhan

Hughes, George Malcolm Jędrzejczyk, Sabina Moir, Ross

Mukhtar, Ridwan Schmieg, Mark

Sparagano, Nicolas Ewan Giovanni Welsh, Cailean

November 9, 2020

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

Testing of the modules developed for the F21DG Group Project can largely be split into three distinct phases: Pre-Integration, Integration and Post-Integration. These phases were developed and agreed upon weeks prior to the testing period, documented in a Testing Plan and discussed with the team. This allowed for a smooth testing phase which could be completed in stages as the different tasks progressed through their development at varying speeds.

Integration, in the context of this document, refers to the testing of interfacing between the inputs/outputs of Task 1, 2 and 3: the Topic Model Generation, the Metrics module and the Visual Report. This does not include the integration between classes within a module. The integration testing was identified as a high risk task, as the latter tasks rely on the serialised model output from Task 1. To mitigate this risk, communication throughout the project was important and a form of continuous testing was completed where models were passed through each stage, where possible, to ensure development stayed largely true to one model.

The need for testing evidence throughout the project was expressed and team members were given directories on the GitLab to place evidence of testing in as it took place throughout development.

1.2 Testing Plan

The Testing Plan defined the scope for each of the three phases and team members were assigned relevant roles weeks ahead of testing taking place.

1.2.1 Pre-Integration

Pre-Integration is defined as all relevant testing required to validate the output of a single task. This primarily referred to unit testing and integration testing of any classes used within a module, or between modules.

Each group had a choice of using software or libraries to assist with unit testing, such as JUnit for Java or Mocha for JavaScript, or to manually test functions/methods. Install testing was also important at this stage to ensure integration testing would be smooth and the task outputs were portable, not only between the team, but for eventual submission as a singular project.

Each team agreed that unit testing should be completed by those not directly responsible for the code being tested, to ensure a higher level of abstraction. The intended purpose of a function/method could be tested with respect to inputs and outputs, without the bias of knowing what exactly the function achieved, or forcing tests to pass with the intended purpose of a function. Whilst each team assigned one person to be responsible for Pre-Integration testing in each task, it was not the sole job of that person to test all code and this allowed for this non-developer-testing approach to be implemented.

1.2.2 Integration

Integration testing involved the outputting of models from Task 1 and ensuring they were accepted by Task 2 and Task 3 modules. Due to the naturally modular nature of the tasks, this was the only real requirement for integration testing as there is no other interaction between modules. As described, this process was intended to be somewhat continuous during development as changes to model serialisation late in development became increasingly time expensive for all groups. It was important to use the full corpus for testing as this phase and a large number of models run to test cases of high disk usage, directory file limits or excessive execution times.

1.2.3 Post-Integration

As with Pre-Integration, this phase consisted of a series of tasks which was the responsibility of a single student who delegated as required. This encompassed system testing, final installation testing and use case testing with the 'customer' or, in this case, the PhD students. Requirements testing, or end-to-end testing, involved testing the program as a singular unit and ensuring that the fully integrated product matched the requirements as described in the Deliverable 1 report. Installation testing was important as it was understood that the final package is likely to be used across a range of operating systems and distributions. The results of this testing fed directly into the user manual, with installation instructions and guides being important for new users. Lastly, it was desired that the PhD students could have a short demonstration of the fully integrated product to identify any glaring issues and make user experience changes to ensure a satisfactory end product.

2 Pre-Integration

2.1 Topic Model Generator

2.1.1 Topic Models

It was necessary to test the custom Topic Model wrapper that holds a generated topic model object and its metrics. Some metric results are dependant upon the provided document corpus or parameter file and, in these cases, tests look for logical cohesion between an objects metrics.

Unit tests were carried out using by Maven using the JUnit library, and gave the following results:

```
{[INFO]} Scanning for projects...
{[INFO]} --------------< com.f21dg.topicModelling:topicModelling >
{[INFO]} Building topicModelling 1.0
{[INFO]} --------------------------------[ jar ]-----------------
{[INFO]} --- maven-surefire-plugin:2.22.1:test (default-test)
@ topicModelling
{[INFO]} -------------------------------------------------------
{[INFO]}  T E S T S
{[INFO]} -------------------------------------------------------
{[INFO]} Running malletWrapper.TopicModelTest
{[INFO]} Tests run: 11, Failures: 0, Errors: 0, Skipped: 0,
Time elapsed: 0.763 s - in malletWrapper.TopicModelTest
{[INFO]}
{[INFO]} Results:
{[INFO]} Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
{[INFO]} -------------------------------------------------------
{[INFO]} BUILD SUCCESS
{[INFO]} -------------------------------------------------------
{[INFO]} Total time:  2.676 s
{[INFO]} Finished at: 2020-11-09T09:06:04Z
{[INFO]} -------------------------------------------------------
```

Test 1
The custom wrapper must be able to store a valid machine learning instances list. All instances in the list are passed through the same Pipe along with the characters and token sequences that are to be featured. This test checks the list is stored locally in the wrapper as once it is passed to MALLET it cannot be retrieved.

Test 2
The custom wrapper must successfully pass the number of topics to be used to MALLET. This tests checks that the default value defaults to 30 if not included in a configuration file and that the value is successfully set if included.

Tests	Pass/Fail
Test 1 (Instances Set)	Pass
Test 2 (Number of Topics)	Pass
Test 3 (Alpha Symmetric)	Pass
Test 4 (Alpha Sum)	Pass
Test 5 (Beta)	Pass
Test 6 (Seed Index)	Pass
Test 7 (Iterations)	Pass
Test 8 (Burn In Period)	Pass
Test 9 (Optimisation Interval)	Pass
Test 10 (Topic Coherence)	Pass
Test 11 (Log-Likelihood < Normalised Log-Likelihood)	Pass

Table 1: Outcome Explanation of Task 1 Functional Requirements

Test 3
This test checks that the model generated is done using a Symmetric or Asymmetric Alpha according to the given flag.

Test 4 and Test 5
The custom wrapper must successfully create a Parallel Topic Model, a custom Topic Model object in MALLET, that has the correct Symmetric Alpha and Beta values set. The default values are 1.0 and 0.01 respectively and this test checks that these values are used if custom values are not included in the configuration file. If the optimisation interval is turned on in the Topic Model wrapper, the alpha value for each topic can be different.

Test 6
MALLET uses a pseudo-random number generator to create the test index. This test checks that the seed for the generator is set when give. If not given, the seed is the computer's real-time clock.

Test 7
This test checks the number of sampling iterations run during a model's training is set in the Topic Generator and in MALLET.

Test 8
This test checks that the correct number of iterations are carried out during MALLET's Topic generation before any hyper parameter optimisation begins.

Test 9
The Topic Generator wrapper must turn on hyper optimisation, to occur every user defined number of iterations. This test checks that this optimisation is started on occurs.

Test 10
This test checks that a coherence is generated for each topic. The validity of the coherence produced was checked during the Metrics testing.

Test 11
This tests checks a NaN is produced when normalised Log Likelihood does not have a Log Likelihood. It also checks the Log Likelihood per number of tokens is less than the model's Log Likelihood.

2.1.2 Model Generation

A series of configuration files were tested against a document corpus in order to validate the data output. Each configuration file sets only a selection of parameters. This tests that the system uses default values even if these parameters are missing. Some invalid parameters were also included to test that the system ignores these parameters. Below are some examples of the configuration files used:

seed_index:120000
alpha_sum:1.0
beta:2.0
iterations:20,21,23

alpha_sum:1.0
beta:2.0
iterations:20,21,23
bingo:2 *NB: Invalid Parameter*

seed_index:100
alpha_sum:1.0
beta:2.0
iterations:20,21,23

seed_index:120000
alpha_sum: 1.0
beta:2.0
iterations: 20,21,23

This allowed for models to be generated and checked manually during the Metric calculations (2.2) to be sure that the JSON output had the expected structure and that the metrics were not showing any inconsistencies. It also gave example outputs that could be used during testing of the Metrics calculations.

2.2 Metrics

2.2.1 Configuration Parsing

It was necessary to test the parsing of the configuration file to ensure parameters could be passed to the metrics without issue.

During set up of this test, a configuration file was created and populated as below. This file was then read by the configuration parser with each of the following tests ensured that the information in this file was recovered. The results are shown below in Figure 1.

```
optionString: Message
```

```
optionDash: a-
optionUnderscore: a_
optionColon: a:
optionSlash: a/
optionDirForwardSlash: C:/Users/Workspace/test_directory/file-1.txt
optionDirBackwardSlash: C:\\Users\\Workspace\\test_directory\\file-1.txt

optionInt: 75
optionDouble: 1.001

option_underscore: 1
option-dash: 1
optionNull:
optionAfterNull: 1

optionBooleanTrue: true
optionBooleanFalse: false
```

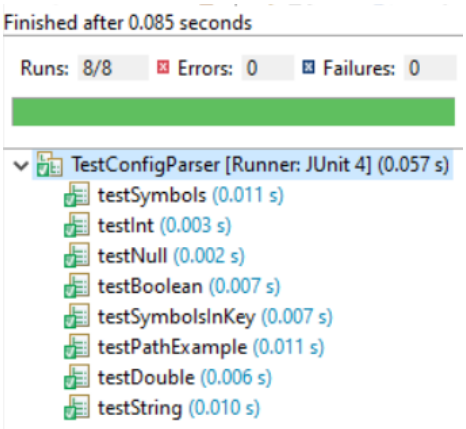

Figure 1: Config Parsing JUnit Test Results

String A configuration must be able to parse and store a string with basic alphabetic characters. *optionString* was retrieved from the configuration and it was asserted that this returned value was *Message*.

Integer A configuration must be able to parse and store a integer. *optionInt* was retrieved from the configuration and it was asserted that this returned value was 75.

Double A configuration must be able to parse and store a double value. *optionDouble* was retrieved from the configuration and it was asserted that this returned value was 1.001.

Boolean A configuration must be able to parse and store a boolean value. *optionBooleanTrue* and *optionBooleanFalse* were retrieved from the configuration and it was asserted that these returned the appropriate boolean values.

Symbols A configuration must be able to parse and store strings which contain symbols certain symbols including: -, _, :, /. *optionDash*, *optionUnderscore*, *optionColon*, and *optionSlash* were retrieved from the configuration and it was asserted that these were the appropriate string.

Null (Empty) A configuration must be able to parse a parameter that has no assigned value, parsing it as null but including it in the configuration nonetheless. It must also continue to parse the next lines after this without any interruption. *optionNull* was asserted to return null and *optionAfterNull* was also tested, simply asserting that its value was one.

Path Strings A configuration must be able to parse a parameter that is assigned to a string that describes a file path. As such, the necessary symbols were explored and compiled into an example path string which used all cases. This extends the earlier symbols test, but confirms that an entire string remains intact. Path strings can have double backward slashes, but not single slashes as Java interprets these as special characters.

2.2.2 Pairwise-Difference Testing

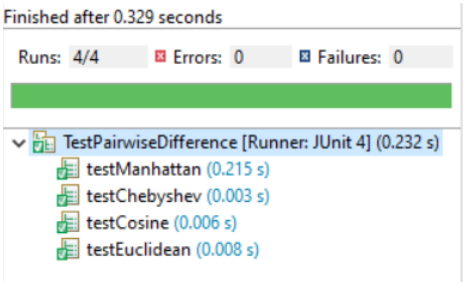

Figure 2: Pairwise Difference JUnit Test Results

Testing pairwise difference required making three simple topic models. The first two models *Model 1* and *Model 2* were identical, while *Model 3* differed in topic 0, word 0. The word weights are shown in the row below a word label and the normalised weights are shown one row below that. A test was performed for each distance type by manually calculating the pairwise difference for these above models and writing unit tests to assert the pairwise-difference metric class matched these hand calculated values. For each manual calculation, a distance matrix was built like the following, replacing δ with the appropriate function. The smallest pairing was extracted from the matrix by looking at the diagonals (since a 2×2 matrix only has two unique pairings).

$$D = \begin{bmatrix} \delta(M_1T_0, M_3T_0) & \delta(M_1T_1, M_3T_0) \\ \delta(M_1T_0, M_3T_1) & \delta(M_1T_1, M_3T_1) \end{bmatrix}$$

Model 1			
Topic 0		Topic 1	
Word 0	Word 1	Word 0	Word 1
1	2	2	1
1/3	2/3	2/3	1/3

Model 3			
Topic 0		Topic 1	
Word 0	Word 1	Word 0	Word 1
2	2	2	1
1/2	1/2	2/3	1/3

Table 2: Weights and normalised weights for test models 1 and 3

Manhattan Manhattan distance was calculated by:

$$\delta(A, B) = \sum_{i=1}^{n} |A_i - B_n|$$

This resulted in the distance matrix:

$$D = \begin{bmatrix} |\frac{1}{3} - \frac{1}{2}| + |\frac{2}{3} - \frac{1}{2}| & |\frac{2}{3} - \frac{1}{2}| + |\frac{1}{3} - \frac{1}{2}| \\ |\frac{1}{3} - \frac{2}{3}| + |\frac{2}{3} - \frac{1}{3}| & |\frac{2}{3} - \frac{2}{3}| + |\frac{1}{3} - \frac{1}{3}| \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & 0 \end{bmatrix}$$

The smallest assignment here is the distance between *Model 1* Topic 0 and *Model 3* Topic 0 and the distance between *Model 1* Topic 1 and *Model 3* Topic 1. Therefore $d = \frac{1}{3} + 0 = \frac{1}{3}$ is the expected output of the metric class.

Euclidean Euclidean distance was calculated by:

$$\delta(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_n)^2}$$

This resulted in the distance matrix:

$$D = \begin{bmatrix} \sqrt{(\frac{1}{3} - \frac{1}{2})^2 + (\frac{2}{3} - \frac{1}{2})^2} & \sqrt{(\frac{2}{3} - \frac{1}{2})^2 + (\frac{1}{3} - \frac{1}{2})^2} \\ \sqrt{(\frac{1}{3} - \frac{2}{3})^2 + (\frac{2}{3} - \frac{1}{3})^2} & \sqrt{(\frac{2}{3} - \frac{2}{3})^2 + (\frac{1}{3} - \frac{1}{3})^2} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{6} & \frac{\sqrt{2}}{6} \\ \frac{\sqrt{2}}{3} & 0 \end{bmatrix}$$

The smallest assignment here is the distance between *Model 1* Topic 0 and *Model 3* Topic 0 and the distance between *Model 1* Topic 1 and *Model 3* Topic 1. Therefore $d = \frac{\sqrt{2}}{6} + 0 = \frac{\sqrt{2}}{6}$ is the expected output of the metric class.

Chebyshev Chebyshev distance was calculated by:

$$\delta(A, B) = \max_{i}^{n} |A_i - B_n|$$

This resulted in the distance matrix:

$$D = \begin{bmatrix} max(|\frac{1}{3} - \frac{1}{2}|, |\frac{2}{3} - \frac{1}{2}|) & max(|\frac{2}{3} - \frac{1}{2}|, |\frac{1}{3} - \frac{1}{2}|) \\ max(|\frac{1}{3} - \frac{2}{3}|, |\frac{2}{3} - \frac{1}{3}|) & max(|\frac{2}{3} - \frac{2}{3}|, |\frac{1}{3} - \frac{1}{3}|) \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & 0 \end{bmatrix}$$

The smallest assignment here is the distance between *Model 1* Topic 0 and *Model 3* Topic 0 and the distance between *Model 1* Topic 1 and *Model 3* Topic 1. Therefore $d = \frac{1}{6} + 0 = \frac{1}{6}$ is the expected output of the metric class.

Cosine Cosine distance was calculated by:

$$\delta(A, B) = 1 - \frac{A \cdot B}{\|A\|\|B\|} = 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

This resulted in the distance matrix:

$$D = \begin{bmatrix} 1 - \frac{(\frac{1}{2}\times\frac{1}{2})+(\frac{2}{3}\times\frac{1}{2})}{\sqrt{(\frac{1}{3})^2+(\frac{2}{3})^2}\cdot\sqrt{(\frac{1}{2})^2+(\frac{1}{2})^2}} & 1 - \frac{(\frac{2}{3}\times\frac{1}{2})+(\frac{1}{3}\times\frac{1}{2})}{\sqrt{(\frac{2}{3})^2+(\frac{1}{3})^2}\cdot\sqrt{(\frac{1}{2})^2+(\frac{1}{2})^2}} \\ 1 - \frac{(\frac{1}{3}\times\frac{2}{3})+(\frac{2}{3}\times\frac{1}{3})}{\sqrt{(\frac{1}{3})^2+(\frac{2}{3})^2}\cdot\sqrt{(\frac{2}{3})^2+(\frac{1}{3})^2}} & 1 - \frac{(\frac{2}{3}\times\frac{2}{3})+(\frac{1}{3}\times\frac{1}{3})}{\sqrt{(\frac{2}{3})^2+(\frac{1}{3})^2}\cdot\sqrt{(\frac{2}{3})^2+(\frac{1}{3})^2}} \end{bmatrix} = \begin{bmatrix} \frac{10-3\sqrt{10}}{10} & \frac{10-3\sqrt{10}}{10} \\ 0 & 0 \end{bmatrix}$$

The smallest assignment here is the distance between *Model 1* Topic 0 and *Model 3* Topic 0 and the distance between *Model 1* Topic 1 and *Model 3* Topic 1. Therefore $d = \frac{10-3\sqrt{10}}{10} + 0 = \frac{10-3\sqrt{10}}{10}$ is the expected output of the metric class.

2.2.3 Resource Management Testing

	Metric				
Number of Topics	Pairwise difference	Perplexity	Topical Alignment	Statistics	Total time (in ms)
0	0	13	11	0	24
1	0	141	26	1	168
2	86	226	45	1	358
10	1870	760	260	2	2892
20	6887	1497	1236	4	9624
40	29878	2891	10029	8	42806
60	66439	4531	32057	12	103039
80	126104	5835	65732	14	197685
100	201772	7181	135108	17	344078
200	801591	14247	853429	39	1669306

Table 3: Resource management testing, all the times are in milliseconds and only one corpus was used.

The metric program was run 11 times, during each iteration the number of models to be processed was increased. It can be observed that the total time keeps increasing every time extra metrics were added. Moreover, it can noted that the times for Pairwise difference and Topical Alignment increase exponentially the more models are added.

2.2.4 Topical Alignment Testing

Topical alignment testing used three hand crafted models. These were used within the topical alignment test and four steps were followed to calculate the correct result.

Step 1: Top M words
The first task of topical alignment is to find the top M words of each topic within each model. M is defined by the user and for testing purposes, the value 3 for M was chosen. The following table showcases the top 3 words found within each topic.

	Top 3 Words		
Topics	Topic Model 1	Topic Model 2	Topic Model 3
0	Cat(10) Dog(9) Chicken(7)	Xaml(11) Clojure(10) C(7)	Aconcagua(9) K2(7) Everest(6)
1	Mercedes(8) Audi(8) Renault(6)	France(8) Iceland(8) Norway(8)	June(9) July(7) May(6)
2	Mig(9) Cessna(8) Dassault(7)	Kooxbaal(9) Mtgambier(7) Sacactun(7)	Pomegranate(8) Blueberry(7) Kiwi(6)

Table 4: Word counts of top M words

Step 2: Normalisation of top M words
After the top M words are found, the sum of the word weights in a given topic are calculated.

	Sum of Topic Words		
Topics	Topic Model 1	Topic Model 2	Topic Model 3
0	39	47	38
1	35	49	34
2	42	45	38

Table 5: Sum of the word weights of each topic

Using these values, the top M word is divided by the total weight of the topic to give a normalised weight for the given word. The calculations that were used can be seen in the table below.

	Normalised Words		
Topics	Topic Model 1	Topic Model 2	Topic Model 3
0	Cat(10/39) Dog(9/39) Chicken(7/39)	Xaml(11/47) Clojure(10/47) C(7/47)	Aconcagua(9/38) K2(7/38) Everest(6/38)
1	Mercedes(8/35) Audi(8/35) Renault(6/35)	France(8/49) Iceland(8/49) Norway(8/49)	June(9/34) July(7/34) May(6/34)
2	Mig(9/42) Cessna(8/42) Dassault(7/42)	Kooxbaal(9/45) Mtgambier(7/45) Sacactun(7/45)	Pomegranate(8/38) Blueberry(7/38) Kiwi(6/38)

Table 6: Sum of topic words across three models

Step 3: Generation of distance matrices
To develop a distance matrix, two models are compared to each other. The value within each cell is calculated using the following equation:

$$Distance = \left|\left(\sum M_x T_n TopMNormalisedWords - \sum M_y T_n TopMNormalisedWords\right)\right|$$

Where M is topic model and T is topic.

Since three models are being used for this test, 3 distance matrices are created. These are: TM1xTM2, TM1xTM3 and TM2xTM3 where TM represents Topic Model. Using the equation above, the following distance matrices are created:

		Topic Model 2		
	Topics	0	1	2
Topic Model 1	0	0.0709	0.1768	0.1555
	1	0.0328	0.1387	0.1174
	2	0.0243	0.0816	0.0603

Table 7: Distance Matrix of Topic Model 1 x Topic Model 2

		Topic Model 3		
	Topics	0	1	2
Topic Model 1	0	0.0877	0.0196	0.1140
	1	0.0496	0.0184	0.0759
	2	0.0075	0.0756	0.0187

Table 8: Distance Matrix of Topic Model 1 x Topic Model 3

Step 4: Agglomerative clustering
To create a cluster, the shortest distance within the distance matrices is looked up and a value of -1 is placed in the cell found. There are three evaluation clauses:

		Topic Model 3		
	Topics	0	1	2
Topic Model 2	0	0.0167	0.0513	0.0431
	1	0.0891	0.1572	0.0628
	2	0.0678	0.1359	0.0415

Table 9: Distance Matrix of Topic Model 2 x Topic Model 3

- If both topics have not been assigned to clusters yet, then they are joined together. The whole row and column containing the shortest distance found receive the value -1, to avoid adding topics from the same model to the same cluster.

- If one topic (A) belongs to a cluster whereas the other topic (B) does not belong to any cluster, then it is checked whether the model of topic B exists in the cluster containing topic A. If it does then the row and column of the distance receive the value -1, to avoid any topics of the model of topic B, being matched again with topic A. If the model of topic B is not contained within the cluster containing topic A, then they are joined together in a cluster. If they join together, the intersection column and row of all the other topics within the cluster and topic B will have their value set to -1.

- If topic (A) belongs to a cluster and topic (B) also belongs to a cluster, then it is checked whether they have common models. If they do then the intersection cell between all topics of both clusters receive the value -1. Only the intersection cell is to change value, since the cluster containing A may have models that are not within the one containing B. Such models should be able to join the cluster with B, but the topics within cluster containing A should never be allowed to join the cluster containing B, since every topic can only be contained within one cluster.

  If no models match between the two clusters, then both are combined into a cluster and all intersection rows and columns between the topics of both clusters receive the value -1, since no topic from a model contained within the cluster should be allowed to join.

The individual steps to reach the result are not shown, since every step follows the three evaluation clauses mentioned above. At the end of the procedure all the cells in every distance matrix contain the value -1.
The clusters calculated are shown below:

	Topic Clusters	
Cluster	Topic Model	Topic
1	1	2
	3	0
	2	0
2	1	1
	3	1
	2	1
3	2	2
	3	2
	1	0

Table 10: The structure of the topic model clusters

This corresponds to the clusters generated from the topical alignment metric and as a result, the test passes.

2.2.5 Perplexity

Testing perplexity was done through the use of multiple JUnit tests in Java, with a custom testing configuration file, three custom testing models and a custom corpus. The perplexity metric has three parameters used throughout the calculation and as with most metrics it appends it's output to a given topic model, such that the model will have a metric "perplexity" with the value of a list of that model's topic's and their associated perplexities. The input is a list of models and a configuration. The following models are used during the testing of perplexity:

testModel1	testModel2	testModel3
`{ "Topics": {`	`{ "Topics": {`	`{ "Topics": {`
`"topic0": {`	`"topic0": {`	`"topic0": {`
`"word0": 1,`	`"word0": 1,`	`"word0": 1,`
`"word1": 1`	`"word1": 2`	`"word1": 3`
`},`	`},`	`},`
`"topic1": {`	`"topic1": {`	`"topic1": {`
`"word0": 1,`	`"word0": 2,`	`"word0": 3,`
`"word1": 1`	`"word1": 1`	`"word1": 1`
`}`	`}`	`}`
`},`	`},`	`},`
`"ID": "0"}`	`"ID": "0"}`	`"ID": "0"}`

Table 11: The attribute of the topic models used for testing

The configuration file used for testing the perplexity metric is a text file containing:

```
perplexity-corpus_path: perplexityTesting/perplexityTestCorpus.txt
perplexity-log_base: 2.0
perplexity-weight_smoothing: 0.00001
```

The corpus at the corpus path given in the configuration is a text file containing:

```
0 en word0 word1 word1 word0 word2
1 en word2 word1 word2 word1 word1
```

Due to perplexity being a purely mathematical value, correct behaviour can be tested through comparing if output values from functions are equal to the mathematically derived correct values. As the test models, the given parameters in the configuration, and the corpus are all fairly simple, the correct mathematical values that should be expected throughout the perplexity metric, can be easily calculated by hand. These expected values can then be compared to the outputs of the constituent functions within *perplexity.class* to verify that the metric is functioning correctly. If the values are the same then the tests pass, any other value is incorrect and so the test fails.

Perplexity is calculated for a given topic through the following process, starting with the input list of models, each with a list of topics, each with a list of words and their associated word counts.

1. Load the configuration and initialise perplexity parameters.
    - This loads the configuration into the program using a given configuration class and then loading the relevant parameter values from that configuration into global variables within the perplexity metric class.
2. Read the corpus from file into the program
    - This loads a corpus textfile into the program using an input filepath and then reading the lines of that file into an array of strings, with each string representing a line from the read file.
3. Normalise word counts into word probabilities.
    - This loads a topic model and normalises all the word counts for each topic into word probabilities.
4. Calculate logarithmic probability of each word in the corpus
    - This takes a probability-normalised topic and a given single-word string, and determines the logarithmic probability of that word appearing, given the probabilities defined within that topic.
5. Calculate logarithmic probability of each sentence in the corpus
    - This takes a probability-normalised topic and a string array of words, and determines the logarithmic probability of the entire sentence appearing, given the probabilities defined within that topic.
6. Calculate logarithmic probability of the entire corpus
    - This takes a probability-normalised topic and a string array of sentences, and determines the sum of the logarithmic probabilities of those sentences, given the probabilities defined within that topic.
7. Calculate topic perplexity of a given corpus

- This takes a probability-normalised topic and a string array of sentences, and determines the mathematical perplexity of that topic given that corpus.

8. Add calculated topic perplexities to the given list of topic models

    - This loads a topic model and calculates the perplexities for each topic given a corpus of words, and then adds those values to the topic model object as an attribute under "Metrics" called "Perplexity" with a value of a list of topics and their associated perplexities.

Each of these stages to the perplexity metric represents a unique unit test that is implemented in Java using JUnit. The JUnit tests for perplexity are all within one file, TestPerplexity.java , which can be found amongst the other source files for Task 2. If all stages to this metric pass their respective unit test, then the perplexity metric itself passes all necessary criteria as a result of it's constituent parts being all valid.

Load the configuration and initialise perplexity parameters Tested through the JUnit test loadConfigParameters() . This is tested by ensuring that the perplexity metric can handle a variety of valid and invalid configuration files, as well as valid and invalid parameters within a given configuration. The only valid input is an object with the type of our custom "Configuration" class. Additionally, the perplexity metric's global variables, which are initialised with hard-coded default values, should be overwritten with any valid corresponding perplexity parameter that is specified in the read configuration file. If the configuration specifies no parameters for perplexity or invalid parameters, then the global variables should remain set to their respective default values and perplexity should execute as normal with those default values. Invalid parameters are any that are not of the correct type, such as a numerical value being given a textual value.

Read the corpus from file into the program Tested through the JUnit test readCorpus() . This is tested by ensuring that the perplexity metric can handle a variety of valid and invalid filepaths. The only valid input is a valid filepath, anything else should raise a "FileNotFound" exception which is handled and outputs an error message to the console whilst skipping over perplexity without affecting the rest of the program. The only valid file, pointed to by the filepath above, is a text file where each line represents a list of words from a document, the start of each line must be a document number, followed by a tab character, followed by the languages of the document, followed by a space-delimited list of words. Any other format should raise an exception which is handled by outputting an appropriate error message to console whilst skipping over perplexity calculation without affecting the rest of the program.

Normalise word counts into word probabilities Tested through the JUnit test normalisationOfTopics() . This is tested by verifying that, when the perplexity metric normalises a variety of test topic models, their topic models now contain the correct values for word probabilities. The only valid values are the exact mathematical numerical values for the normalisation of the word

counts within the given test models. This means that in a normalised topic model, for every topic, a given word should have an associated value that is exactly the word count divided by the sum of all word counts for that topic.

Calculate logarithmic probability of each word in the corpus Tested through the JUnit test wordProbability() . This is tested by verifying that, when the logarithmic probability of a given word is calculated in the perplexity metric, the correct value is generated. There are only two valid values: if the word is contained within the topic then the only valid output is the mathematical value $logB(x)$, if the word is not contained within the topic, then the only valid output is the mathematical value $logB(S)$. Where B is the parameter for the logarithmic base, x is the topic's probability for the given word, and S is the parameter for the smoothing value.

Calculate logarithmic probability of each sentence in the corpus Tested through the JUnit test sentenceProbability() . This is tested by verifying that, when the logarithmic probability of a given sentence is calculated in the perplexity metric, the correct value is generated. The only valid value is the sum of the logarithmic word probabilities within that sentence.

Calculate logarithmic probability of the entire corpus Tested through the JUnit test corpusProbability() . This is tested by verifying that, when the logarithmic probability of a given corpus is calculated in the perplexity metric, the correct value is generated. The only valid value is the sum of the logarithmic word probabilities of all sentences within the given corpus.

Calculate topic perplexity of a given corpus Tested through the JUnit tests corpusWordCount() , topicPerplexities() , and equalPerplexities() . This is tested by verifying that the correct perplexity value is generated for a given topic and given corpus. This includes testing that the correct value is calculated for the total number of words in the given corpus, and ensuring that the calculated perplexity is stable by verifying that calculated perplexities for identical topics on the same corpus output the same perplexity value. The only valid value is the numerical result of the formula, $B^{-(x/n)}$. Where B is the parameter for the logarithmic base, x is the sum of the topic's probability for every sentence in the given corpus, and n is the total number of words in the corpus.

Add calculated topic perplexities to the given list of topic models Tested through the JUnit tests addMetricToModels() . This is tested by verifying that when the perplexity metric is successfully run on a set of given topic models, every model has a new attribute added under the "Metric attribute called "Perplexity". This test only passes if every model output by the perplexity metric has the new "Perplexity" attribute.

All the unit tests above are implemented by respective JUnit tests, the execution of which can be seen below:

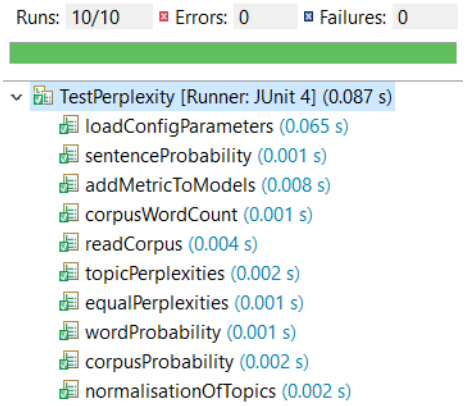

Figure 3: Perplexity metric JUnit Test Results

2.2.6 Topic Model

The topic model testing was completed using JUnit tests. The first type of tests conducted were standard models being used and checked to see if the topic model stored the correct values. The same was also done on scenarios where entire dictionaries were missing (e.g. the metrics dictionary not being in a model) and seeing if this caused any issues. Finally, adding metrics to the metrics dictionary under different scenarios were also conducted.

The first test conducted was to see if an input model that contained no metric dictionary would still be valid. This passed since in the event that no metric dictionary exists in the inputted model, an empty map is created in it's place to allow for metrics to still be added to a newly created dictionary.

The second test was based on having no parameters. Although currently, parameters aren't used within metrics and a crash wouldn't occur, it is still instantiated incase these are used within the program in the future.

The third test looked at having no topics dictionary being present in the inputted model. This is also instantiated so that metrics that grab the dictionary and analyse the topics wouldn't crash the program.

The fourth test used an empty dictionary ({}), containing no information. The program still runs since everything needed from a normal dictionary is instantiated and will still allow for topic model class to be in a state where it can be ran on metrics.

The fifth test checked whether or not a metric could be added when an input model that was empty ({}). This passed since a new metric dictionary was created to store the information.

The final test dealt with the standard scenario where all dictionaries were present and had values. It checked to see if the expected input was stored correctly within the topic model class.

All tests passed and allowed the program to fully complete it's execution without causing any crashes.

2.3 Visual Report

The Visual Report was tested with several test cases to make sure that all the requirements implemented were functioning as expected. Also tests were done to make sure that the output being generated on the scatter plot was as expected while performing the tasks which a user would carry out while interacting with the Visual Report. The following test cases were looked at to check the functionality of the Visual Report:

Expected Results

T-1: The page should visualise two scatter plots showing the correlation of two metrics depending on the x and y axis from the models present in data-files directory. The x and y axis should be different for the two scatter plots and the z axis should not be set to a metric.

T-2: The data on the scatter plot should be updated with the models found in the new directory.

T-3: The scatter plot should be updated to show the correlation between the new metrics selected and the values of the axis should be updated. The second scatter plot should remain the same.

T-4: The configuration file from the Topic Model Generator is displayed under the scatter plots showing the parameters and their values used to generate the models.

T-5: The user should be able to select up to 3 models to display the data and the word clouds for. The data of the models selected should be displayed under the Selected Model 1, 2 or 3 tab depending on how many models have already been selected.

T-6: The first model selected by the user should have its colour updated to blue on the scatter plot, second model updated to red and third model updated to green.

T-7: When a user selects an already selected model that model's data should be removed from the selected model tab under the scatter plots and the plot on the scatter plot should be updated back to yellow.

T-8: After deselecting a model, the user should be able to select another model to display under the Selected Model's tab.

T-9: After selecting a model, a tab should be displayed under the Selected Model which allows the user to see all the data related to that model such as parameters used, metrics calculated and the topics and their words along with the weights.

T-10: After a model has been selected, its word bubbles should be shown under the selected models' tab.

T-11: After a user selects a model to view, its colour changes to either blue, red or green depending on the number of previously topics selected. Along with that the colour of that model should also be updated on the second scatter plot.

ID	Description	Pass/Fail
T-1	Test access to the Visual Report	Pass
T-2	Test changing the directory where the Model(s) are stored	Pass
T-3	Test changing the x, y or z axis on the first scatter plot	Pass
T-4	Test user is able to see the parameters used to generate the models	Pass
T-5	Test selection of the Model(s)	Pass
T-6	Test User is able to see correct Model	Pass
T-7	Test deselecting a Model	Pass
T-8	Test selection of another model after deselection	Pass
T-9	Test the user is able to see the selected model's data	Pass
T-10	Test visualisation of the word bubbles	Pass
T-11	Test selection of a Model on one scatter plot	Pass

Table 12: A list of tests and their success status

Several models were generated from the Topic Model Generator and the Metrics Program and those models were displayed on the scatter plot to make sure that the Visual Report was able to handle data generated from different models using various parameters. The metrics were also compared against the metrics in the JSON files to make sure that the outcome was the same as the one found in the files. For the word clouds, the words and the font sizes of the words were compared with the data found in the JSON files for selected models to make sure that the values matched. The data taken from the JSON files was also shown on the console to make sure that valid data was appended in the array. The data used for the generation of the word bubbles was also outputted to the console and the words and their font sizes were compared to make sure that they matched.

2.3.1 Pearson's Correlation Coefficient

A total of 3 models were used where the metric, log likelihood was loaded in the x-axis and coherence stats mean was loaded in the y axis.

The values of x axis were **-8, -2 and -3** and the values of y-axis were **-7, -6 and -9** respectively. The mean of x-axis was calculated to be **-4.3** and the mean of y-axis was calculated to be **-7.3**.

To get the numerator the equation used was: **(xAxis - xMean) * (yAxis - yMean)**. This equation was ran for the number of models which was three and in the end the numerator was calculated to be **-0.33**.

To get the denominator values for x the following formula was used: **(xAxis - xMean)**^{2} for each model and then the values were added up.

The final xDenominator value was **20.67**. The same was done to get the

denominator value for yAxis **(yAxis - yMean)**^{2} which turned out to be **4.67**.

To get the final value, the two denominators were added up and square root of the value was taken giving the result **9.82**. Finally, to get the Pearson's r value the numerator was divided by the denominator and the final value of **-0.034** was acquired which was the exact same as the value that the function outputted which can be seen below.

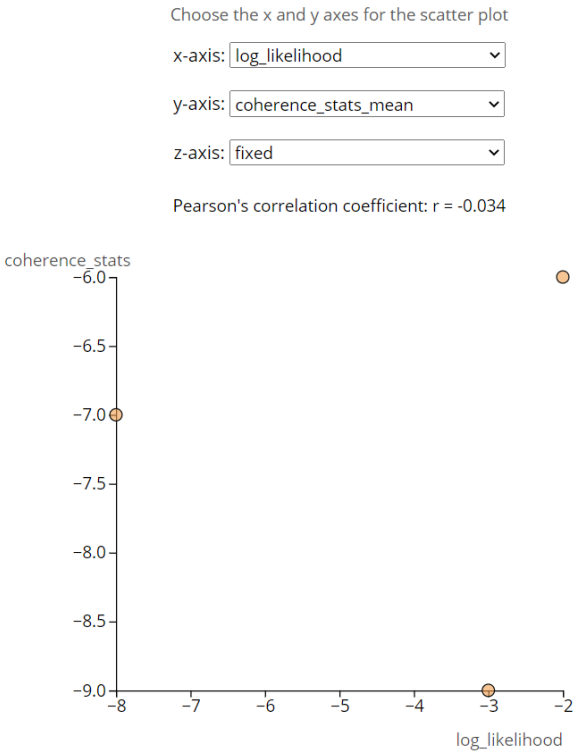

Figure 4: Pearson's Correlation Coefficient

3 Integration

Since the factor which integrated the three sections of the software was the serialised model, integration testing was mostly carried out by rolling out a collection of topic models each time a change was made and letting the individual teams deal with them in the way they chose - guided by integration meetings and discussion during team meetings. As such, there is little evidence to report beyond the backlog of sample models on the development repository.

3.1 Serialised Model

Integration testing was carried out continuously during the concurrent development of all three sections (namely model generation, metric calculation and the visual report) by providing serialised models which reflected the design choices made during serialisation of the first two sections.

It was particularly critical to ensure continuity of the serialised design since a single change in one section would propagate and inevitably break another section - interfering with development. As such, any proposed changes which were brought up in the meeting and agreed upon were standardised into the model afterwards and a batch of models were produced which reflected this change.

After any change was made, each team would be asked to report on any problems they may have had when incorporating the design of the new models.

3.1.1 Fake Model Generator

During the early stages of the project, a serialised model design was conceived, however since the model generation program had not yet been put together, a script was written in Python to produce a collection of *fake* sample models which the teams were able to use for appending metrics and displaying.

Models were generated with randomised values:

- Random parameters were included to demonstrate model design.
- Random topics complete with real words (from a short dictionary) and randomised weights for each were included to allow metric evaluation to be performed.
- Random metrics were were included to allow development on the visual report to begin.

4 Post-Integration

4.1 End to End Testing

End to End testing was carried out to make sure that all the implemented requirements would function as required when a user would be interacting with the system. The program was ran from start to end and all the requirements were evaluated against the output that was produced with the expected output. A brief explanation of how each requirement has been met can be found below:

Task 1 Functional Requirements

ID	Outcome	Priority
FR-1	Achieved	MUST
FR-1.1	Achieved	MUST
FR-1.2	Achieved	MUST
FR-1.3	Achieved	MUST
FR-1.4	Achieved	MUST
FR-1.5	Achieved	SHOULD
FR-1.6	Achieved	SHOULD
FR-1.6.1	Partially Achieved	SHOULD
FR-1.7	Achieved	COULD
FR-1.8	Not Achieved	WANT

Table 13: Outcome Explanation of Task 1 Functional Requirements

FR-1: The topic model when ran with a given corpus and parameters file, produces topic models successfully which can be seen in the output folder.

FR-1.1: The software is a CLI and is able to run successfully given the correct command along with the required parameters.

FR-1.2: The configuration file contains the different parameters that will be used for generating the models, which can all be changed by the user to their liking. Along with that some optional parameters are also available for the users to change if default values are not suitable.

FR-1.3: The Topic Model Generator successfully outputs the number of models depending on the combination of parameters specified.

FR-1.4: All the models are serializable and stored in a output directory .

FR-1.5: The Topic Model Generator is able to run successfully and generate a single model if provided with a single input.

FR-1.6: The configuration file is in ASCII format and is imported to define the system's configurations.

FR-1.6.1: This requirement has not been fully achieved as the configuration file is not fully modular and expandable.

FR-1.7: A new Topic Modelling option has been created as a wrapper to the existing system.

FR-1.8: The system is unable to compute multiple topic models in parallel.

Task 2 Functional Requirements

ID	Outcome	Priority
FR-2	Achieved	MUST
FR-2.1	Achieved	MUST
FR-2.2	Achieved	MUST
FR-2.3	Achieved	MUST
FR-2.4	Achieved	MUST
FR-2.5	Achieved	MUST
FR-2.6	Achieved	MUST
FR-2.6.1	Achieved	SHOULD
FR-2.6.2	Achieved	COULD
FR-2.7	Not Achieved	COULD
FR-2.8	Achieved	MUST
FR-2.9	Not Achieved	WANT

Table 14: Outcome Explanation of Task 2 Functional Requirements

FR-2: The Metrics Program is able to successfully calculate a range of difference/ quality measures which can be seen in the resulting JSON files.

FR-2.1: The Metric Program computes quality measures for each model given to it, the results of which are present in the output JSON files.

FR-2.2: The Metric Program is successful in calculating the difference measures between different models which can be seen in the output JSON files.

FR-2.3: The aggregated results from modules for all topics can be seen in the JSON files produced.

FR-2.4: The resulting JSON files contain all the metrics calculated by the program on the topic models.

FR-2.5: The Metrics program is able to compute log-likelihood successfully.

FR-2.6: The program is able to successfully compute at least one perceptual difference metric which is displayed in the JSON files.

FR-2.6.1: The pair wise difference measure is able to me computed by the program and can be found in the output JSON files.

FR-2.6.2: The topical alignment measure can be computed by the program and can be found in the TopicalAlignment JSON file.

FR-2.7: Topic Size Variability measure has not been implemented in the program due to time constraints.

FR-2.8: The topic coherence metric calculated can be seen in the resulting JSON files.

FR-2.9: Cluster Size Variability measure has not been implemented in the program due to time constraints.

Task 3 Functional Requirements

ID	Outcome	Priority
FR-3	Achieved	MUST
FR-3.1	Achieved	MUST
FR-3.2	Achieved	MUST
FR-3.3	Achieved	SHOULD
FR-3.4	Achieved	SHOULD
FR-3.5	Achieved	SHOULD
FR-3.6	Achieved	COULD
FR-3.7	Achieved	COULD

Table 15: Outcome Explanation of Task 3 Functional Requirements

FR-3: The Visual Report shows the breakdown of data through different methods such as scatterplots and word bubbles.

FR-3.1: The scatterplots show the different quality measures that have been calculated for the topic models.

FR-3.2: The scatterplot shows the connections between the different topics depending on the metric chosen for the x, y and z axis.

FR-3.3: The graphical representation of the topic models is shown through the scatterplots and the word bubbles.

FR-3.4: The different models can be visually compared through the scatterplots and the word bubbles

FR-3.5: The user is able to select the x and y axis for the scatterplots which shows the correlation between the chosen metrics.

FR-3.6: The data used for the Visual Report can be seen by selecting the model and expanding the resulting tabs produced under the scatterplots.

FR-3.7: The topics' top words can be seen through the word bubbles produced.

Task 1 Non - Functional Requirements

ID	Outcome	Priority
NFR-1.1	Achieved	MUST
NFR-1.2	Achieved	MUST
NFR-1.3	Achieved	MUST
NFR-1.3.1	Achieved	MUST
NFR-1.4	Achieved	MUST
NFR-1.5	Achieved	MUST
NFR-1.5.1	Achieved	COULD
NFR-1.6	Partially Achieved	SHOULD
NFR-1.7	Not Achieved	COULD
NFR-1.8	Not Achieved	COULD

Table 16: Outcome Explanation of Task 1 Non-Functional Requirements

NFR-1.1: The system implemented makes uses of the MALLET java package.

NFR-1.2: The system designed has been made modular to increase its maintainability.

NFR-1.3: The system has been made portable so it can run easily on other devices.

NFR-1.3.1: Standardised Java version has been used to build the system

NFR-1.4: Configuration of multiple parameters has been implemented to run in parallel per execution.

NFR-1.5: The system is scalable for further development in the future.

NFR-1.5.1: The system can be executed on any hardware.

NFR-1.6: The program provides error reports, but the error reports are not extensive.

NFR-1.7: The system is unable to produce execution log reports detailing the input parameters and outputs.

NFR-1.8: The system has not been parallelised to decrease the execution time.

Task 2 Non - Functional Requirements

ID	Outcome	Priority
NFR-2.1	Achieved	MUST
NFR-2.2	Achieved	MUST
NFR-2.3	Achieved	MUST
NFR-2.3.1	Achieved	MUST
NFR-2.3.2	Achieved	MUST
NFR-2.3.3	Achieved	SHOULD
NFR-2.4	Not Achieved	COULD

Table 17: Outcome Explanation of Task 2 Non-Functional Requirements

NFR-2.1: The data output file produced is in a standardised JSON format.

FR-2.2: The system has been made modular to allow separation of individual difference/ quality measure.

FR-2.3: Intensive testing has been done to make sure that the system is reliable.

FR-2.3.1: The system is able to handle all known extreme values if provided.

FR-2.3.2: The system is able to handle all know errors and continue with its execution.

FR-2.3.3: The system is able to produce extensive error reports.

FR-2.4: The system is unable to parallelise the execution of measures to increase performance.

Task 3 Non - Functional Requirements

ID	Outcome	Priority
NFR-3.1	Achieved	MUST
NFR-3.1.1	Achieved	MUST
NFR-3.1.2	Achieved	MUST
NFR-3.2	Achieved	MUST
NFR-3.3	Achieved	MUST
NFR-3.4	Achieved	SHOULD
NFR-3.4.1	Achieved	COULD
NFR-3.4.2	Not Achieved	WANT

Table 18: Outcome Explanation of Task 3 Non-Functional Requirements

NFR-3.1: The Visual Report is able to perform on any given platform.

NFR-3.1.1: The Visual Report is able to be accessed from any standard web browser.

NFR-3.1.2: The Visual Report is able to be accessed through any standard OS.

NFR-3.2: The Visual Report has been made modular so additional graphs and logs can be implemented easily.

NFR-3.3: The Visual Report has been developed using d3.js.

NFR-3.4: The user is able to interact with features on the visual report such as scatterplots and tabs for selected models.

NFR-3.4.1: The user is given feedback on the interaction performed by them on the Visual Report.

NFR-3.4.2: The interface is unable to remember any past interactions done by the user on the Visual Report.

4.2 Installation Testing

Installation testing was completed across 3 different operating systems; Windows, Linux and MacOS to make sure that the final version of the software was able to install and execute successfully on different operating systems and produced the expected output results when ran.

4.2.1 Windows

In order to run the program, Java version 11 or above is required. That is because the class files have been compiled with Java version 11 and running the program with a previous Java version will result in a runtime error. The latest Java versions can be downloaded from:
https://www.oracle.com/java/technologies/javase-downloads.html

The Topic Model Generator was ran using the command found below:

```
>java -jar topicModelling-1.21-jar-with-dependencies.jar -c modelCorpus.txt -p params.conf
```

Figure 5: Command For Running Topic Model Generator

The jar file being used was located in the following directory:

f21dg-d2/Task-1/topicModelling/target

For the program to execute, 2 command line parameters are required with 4 optional parameters which can be specified if required. The first required parameter was the corpus file path and the second one was the configuration file path which contained the parameters for the execution of the program. The **params.conf** file that was used had the following parameters:

- seed index: 1,10,100

- alpha sum: 1.0
- symmetric alpha : false
- beta:2.0
- iterations: 20,21,23,24,25
- number of topics: 20
- burn in interval: 1
- optimise interval:2

The following is a screenshot showing part of the execution for the program:

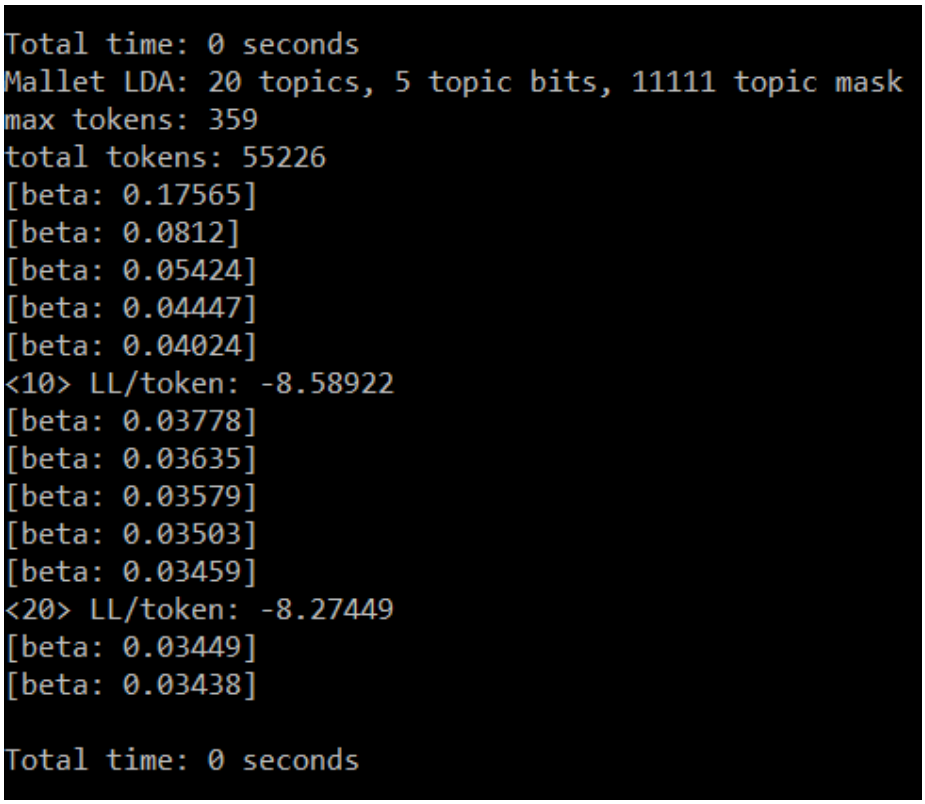

Figure 6: Running of Topic Model Generator

A total of 15 models were generated by the Topic Model Generator which were placed in a new folder in the output sub-directory of target.

The Metrics Program used for calculating the metrics of the models produced was ran with Eclipse IDE (Version 4.17, Version Name: 2020-09) which can be downloaded from **https://www.eclipse.org/downloads/**. The following directory was imported as a project in Eclipse: **f21dg-d2/Task-2/F21DG-Obj2**

The following changes were made to the configuration file:

- model in directory - which was set to the folder of the models generated by the Topic Model Generator
- models out directory - which was set to the Task-3/data-files/NewFolderName from which the Visual Report would read the generated models for displaying the different metrics and other required data
- perplexity-corpus path - which was set to the corpus file path.

The configuration file used can be found below:

```
models_in_directory: C:/wamp64/www/TESTW/f21dg-d2/Task-1/topicModelling/target/output/08-11-20T12_05_43
models_out_directory: C:/wamp64/www/TESTW/f21dg-d2/Task-3/data-files/run

pairwise_difference-top_M:10
sample_metric-messsage: thisismymetricmessage
pairwise_difference-distance_type: taylorlautner

perplexity-log_base: 2.71
perplexity-weight_smoothing: 0.001
perplexity-corpus_path: C:/wamp64/www/TESTW/f21dg-d2/Task-1/topicModelling/target/modelCorpus.txt
```

Figure 7: Configuration File for Metrics Program

Screenshot of program successfully reading topic models.

```
---------LOADING MODELS---------
Loading topic models from: C:\wamp64\www\TESTW\f21dg-d2\Task-1\topicModelling\target\output\08-11-20T12_05_43
Successfully loaded 15 topic models.
```

Figure 8: Reading Models

Screenshot of program outputting models to the specified directory.

```
---------OUTPUT MODELS----------
Writing output models to: C:/wamp64/www/TESTW/f21dg-d2/Task-3/data-files/run/
message.
```

Figure 9: Output Models

For displaying the Visual Report, WampServer (Version: 3.2.0) that stands for Windows, Apache, MySQL and PHP was used which is available at: **https://sourceforge.net/projects/wampserver/files/**. WAMP acts like a virtual server and allows you to run the project files through localhost. The **Task-3** folder was placed in the **www** sub-directory of **wamp64** and the **index.php** file was accessed on the localhost server to display the virtual report using the commands **localhost/Task-3/index.php**.

4.2.2 MacOS

After upgrading the Java version to the latest one in order to avoid the runtime error described during the windows installation, the same command was used

as in the windows installation for the running of the Topic Model Generator. The same configuration file was used as well.
Screenshot showing the execution of the Topic Modelling Generator

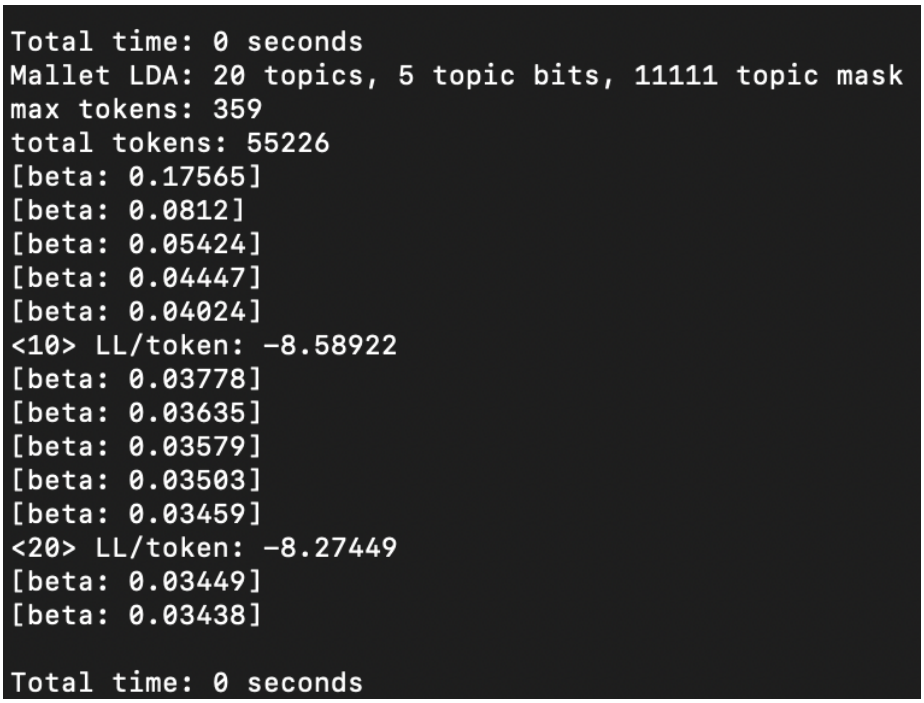

Figure 10: Topic Model Generator Execution

A total of 15 models were generated by the Topic Model Generator which were placed in a new folder in the output sub-directory of target. The Metrics Program used for calculating the metrics of the models produced was ran with Eclipse IDE (Version 4.17, Version Name: 2020-09) which can be downloaded from **https://www.eclipse.org/downloads/**. The directory imported into eclipse was:
f21dg-d2/Task-2/F21DG-Obj2

The configuration file was the same one that was used for the windows installation,in which the only changes made were for model in directory, model out directory and perplexity corpus file paths. They were changed to reflect the updated file paths. The execution of the program was as expected and was the same as described during windows installation.

For the hosting of the visual report, MAMP software (Version: 6.0.1) was downloaded from **https://www.mamp.info/en/downloads/**. MAMP server was used to run the project files on the device. The Task-3 folder was placed in the **htdocs** sub-directory of **MAMP** and after starting the server, the Visual Report was displayed on the web page which was accessed from the following link: **localhost:8888/Task-3/index.php**

4.2.3 Linux

In order to run the Topic Modelling Program, Java version was upgraded to the latest one for the same reasons as described in the Windows and MacOS installation settings. That was done because the class files had been compiled with Java version 11 and running the program with a previous version resulted in a runtime error. The Java versions can be downloaded from: **https://www.oracle.com/java/technologies/javase-downloads.html**

The Topic Model Generator was ran using the same configuration file as the one used during Windows and MacOS installation testing. The command used was the same as well which is shows shown below:
Screenshot of Topic Model Generator Execution Command:

```
java -jar topicModelling-1.21-jar-with-dependencies.jar -c modelCorpus.txt -p params.conf
```

Figure 11: Topic Model Generator Execution Command

The configuration file had the same parameters as the one used for windows testing and no other optional parameters were changed.

For the Metrics Program, Eclipse software (Version 4.14, Version Name: 2019-12) was used. The following directory was imported as a project in Eclipse:
f21dg-d2/Task-2/F21DG-Obj2

The changes made to the configuration file for the Metrics Program were the same as the changes made during Windows and MacOS testing which included changing the model in directory, models out directory and perplexity corpus path to reflect the updated paths of the files. The screenshot below shows the loading of the metric classes in Eclipse:

```
---------LOADING METRICS--------
Loading metrics from: /home/hw/TESTW/f21dg-d2/Task-2/F21DG-Obj2/metrics
Loading metric: PairwiseDifference.class
Loading metric: Perplexity.class
Loading metric: Robert Pattinson.png
Loading metric: TestMetricSample.class
Loading metric: TopicAlignment.class
```

Figure 12: Loading of Metrics in Eclipse

For the display of the Visual Report XAMPP server (Version 7.4.11) was downloaded which is a Apache distribution containing MariaDB, PHP and Perl allowing running of the Visual Report on the local device. It can be downloaded from https://www.apachefriends.org/download.html. After the download had completed and the server had been started, the **Task-3** folder was placed in the **htdocs** sub-directory of **lampp**. Afterwards from a browser of choice, the Visual Report was accessed through **localhost/Task-3/index.php**.

4.3 Use Case Testing

A demonstration was set up with one of the PhD students (Pierre Le Bras) to make sure that our current product satisfied the use cases provided to the development team during the early stages of the project. The demonstration helped the user get familiar with the end product and additional features and changes were recommended throughout the session which would make the system better and increase the satisfaction of the end user.

The first part of the demonstration included execution of the Topic Model Generator to generate several topic models. Along with that the configuration file was also shown and the different parameters that could be changed for the execution of the program were explained. Lastly, the resulting JSON output files showing the topics and their words were also briefly talked about.

For the Topic Model Generation the following suggestions were recommended:

- Since the mandatory and optional parameters for the running of the program were not very clear, it was suggested to display the mandatory parameters at the top of the parameters list or have the parameters list split into two sections, mandatory and optional, to provide more clarity to the user.
- The naming of the output models generated by the module were not very readable and it was suggested to change the naming technique to make them more readable. One way it could be done was by naming the models in ascending order starting from 0 e.g. for 4 output models the names would be model-0, model-1, model-2 and model-3.

The second part of the demonstration included running the Metrics Program to generate the metrics for the models outputted by the Topic Model generator. The resulting JSON files were shown which displayed the metrics for the several topics.

For Metric calculation, the following suggestions were made:

- After the Topic Model Generator has finished running, the user has to manually execute the Metrics Program. It was suggested to write a wrapper around the two separate programs so they execute one after the other provided with a corpus, models configuration file and metrics configuration file. This will save the user from having to manually run the second program.
- Metrics cannot be disabled; it was suggested to have a flag for each metric that allows the user to select which metrics to calculate for the topic models e.g. pairwise-difference-run: TRUE/ FALSE where if set to TRUE would result in the program calculating that metric and if set to FALSE will not calculate that metric.

The demonstration was concluded by presenting the Visual Report using the JSON files from the first two parts of the program.

For the Visual Report the following suggestions were recommended:

- The Visual Report only displays one scatterplot, and it was suggested to add another scatterplot so that further two more metrics can be visualised at the same time.
- It was suggested to reduce the distance between the selection of the axis and the scatterplot since if multiple scatterplots are to be implemented then it might get confusing which axis belongs to which scatterplot.
- It was recommended to display the data from the configuration file which contained the parameters that were used while generating the topic models.
- An implementation of a third (z) axis was proposed which would allow the user to pick a third metric for the visualisation of the scatterplot.
- The number of models that can be selected at one time was to be increased from 2 to 3 as there was unused space on the Report which could be utilised for displaying another model.
- The words in the word bubbles were a bit difficult to read so a change of font was recommended and it was suggested to use a more neutral font such as Open Sans.
- It was suggested to give user some feedback on how many models can be selected at one time and that information would be displayed under the scatterplot.
- The word bubbles were a bit clustered when shown on the Visual Report and it was advised to only show the top 30 words from the topic as that many words should be enough for the user to make up their mind regarding the topic.

Many suggestions that were given were taken into consideration and worked on, but unfortunately some of the changes advised had to be left out due to time constraints.