Verification and Validation Report GMM-EM

Kim Ying WONG

April 18, 2024

1 Revision History

Date	Version	Notes
11 April 2024	1.0	First Draft

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test

Refer to SRS

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

1. Functional requirement R1: Missing values or illegal values will be detected by the software. It has been verified via 4 different unit tests. The details on the unit tests listed on section 5.1. Our software GMM-EM could detect following errors : File error (File not Found), missing values error (empty file) or any values that cannot convert into double will be consider as illegal by our GMM-EM.

2. Functional requirement R2: Convergence should be guaranteed for our algorithm. The GMM-EM generally meets the requirement. It passed all the test cases which show no convergence issues (Detailed listed in section 5.2). Unit tests on calculations for the EM algorithm has been carried and the result has been verified. For those exceptional or unexpected cases (have not been encounted in our test cases yet), an exception for convergence error, which indicates the model cannot converge to a sensible solution with the given iteration, will be displayed to the user.

3. Functional requirement R4: The output will be an array specific the which cluster data points should belong to. A system test is carried out to verify the workflow for our GMM-EM to produce a sensible output. The system test cases mentioned in VnV plan has been implemented. The first ideal test case on two Gaussian distribution produce output that correctly indicate which data point belongs to the correct corresponding distribution. The second practical test case also correctly indicates the cluster that datapoints should belong to.

4 Nonfunctional Requirements Evaluation

4.1 Accuracy

The accuracy is our main concern in our software GMM-EM. As stated at the VnV plan, we decide to use Gaussian Mixture package from Python scikit-learn as our pseudo oracle. In order to compare the accuracy, we use the rand index as our metric. In our original plan, we use adjusted rand index (ARI), but for simplicity, we adopt the easier formula with rand index. In our system test, we test the accuracy with two dataset. We compare the result with current pseudo oracle and find that our model produce the same result. The rand index and accuracy equals 1 which means no miss-classification in our case. The The result are visualized as follows.

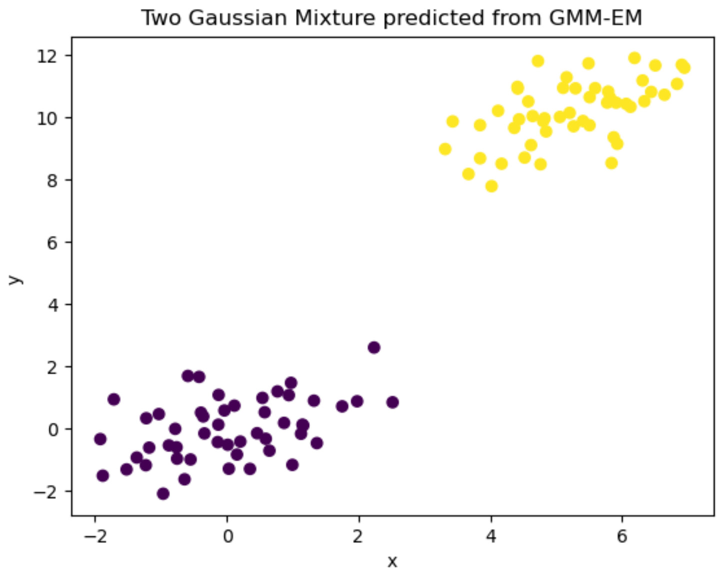

Figure 1: System Test 1: GMM-EM (Two Gaussian)

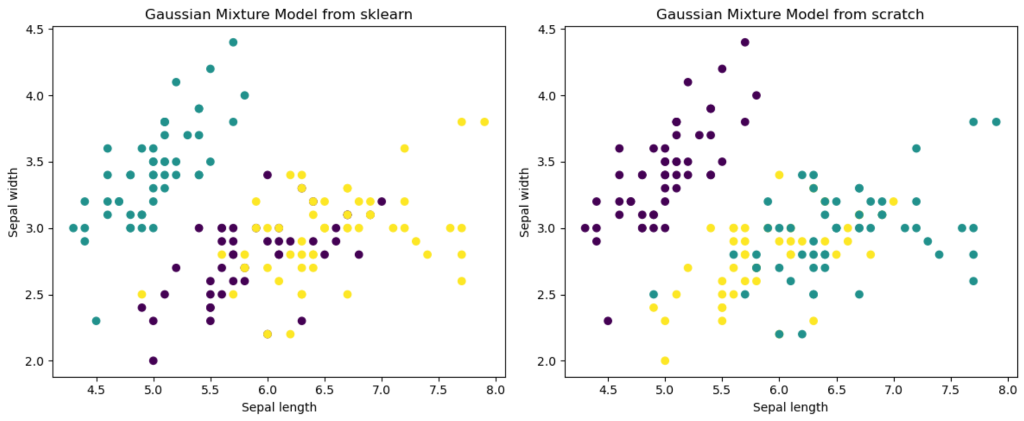

Figure 2: System Test 2: Scikit-learn vs GMM-EM

4.2 Understandability

There is no valid testing for the understandability for the GMM-EM, while few items has been done to ensure the library is understandable for user or developer. Comments in code has been added to every functions and classes.

4.3 Maintainability

There is no valid testing for the Maintainability for the GMM-EM, while in designing every module, we ensure every module only consists of one secret. This priniciple should give a larger flexibility if any changes needed to be made in the future development.

4.4 Performance

In our original plan, we aim at comparable speed to our pseudo oracle. However, in our performance test (System Test 3). It shows that our GMM-EM is much slower than the pseudo oracle. We compare the time usage for a dataset with 300000 datapoints and list the performance difference in Table 1. Our implementation is 5 times slower, but it is still acceptable since our software could be further optimized.

	Running Time
Our GMM-EM	2.78 s
scikit-learn	0.416 s

Table 1: Caption

4.5 Portability

This part relates to our automated testing and due to time being, testing has not been done at this stage. We tried to build a workflow in GitHub Actions which utilize the docker for building the library in different operating systems. However, the test has not been carried out at this stage. The remedy at this stage is using Cmake to configure our building process for the library GMM-EM.

5 Unit Testing

Unit testing for every module has been implemented.

5.1 Input Module

We perform four unit tests for the input module. This could refer to the VnV plan.

1. test-id1-valid-input : Passed
2. test-id2-illegal-values : Passed
3. test-id3-InvalidCSV : Passed
4. test-id4-Empty : Passed

The user could build the library with Cmake and result could refer to the executable named with "InputTest".

5.2 Model Initiation Module

1. test-id5-ClusterEqGtDateSize : Passed

5.3 Model Training Module

1. test-id6-likelihood : Passed
2. test-id7-normal-cal : Passed

5.4 Model Predict Module

1. test-id8-NoFitPrediction : Passed
2. test-id9-PostiveIntPrediction : Passed

5.5 Output Module

1. test-id10-ValidCSV : Passed
2. test-id11-InValidPath : Passed

6 Changes Due to Testing

Exception handling is enhanced after reviewed by tests and reviewer team.

7 Automated Testing

Several attempts in automated testing has been done (through GitHub Action), but due to time being, the testing are eventually done by the developer. Automated testing have not been adopted in this software.

8 Trace to Requirements

	R3	R4	NFR 1	NFR 4
5.1.1		X	X	
5.2.1	X		X	
5.2.2	X			X

Table 2: Traceability table for system test

9 Trace to Modules

M1 (input module)	UT 1	UT 2	UT 3	UT 4
M3 (Initiation module)	UT 5			
M4 (Training module)	UT 6	UT 7		
M5 (Predict module)	UT 8	UT9		
M6 (Output module)	UT 10	UT 11		

Table 3: Traceability Between Test Cases and Modules

10 Code Coverage Metrics

Due to time being, the code coverage tools has not been adopted at current stages, but are eventually done by the developer.