System Verification and Validation Plan for GMM-EM

WONG, Kim Ying

April 18, 2024

Revision History

Date	Version	Notes
15 Feb 2024	1.0	First Draft
19 Feb 2024	2.0	Second Draft

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
UT	Unit Test

symbols, abbreviations, or acronyms — you can simply reference the SRS SRS if appropriate

2 Introduction

This VnV document provides an introductory blurb and roadmap of the Verification and Validation plan. This consists of general information of GMM-EM, verification objectives, plans , tools for testing, description on system test for both functional requirements and non-functional requirements and units test(to be filled later).

3 General Information

This part introduce the general information for our software GMM-EM

3.1 Summary

The software GMM-EM is being tested. The fundamental function of this software is to solve an unsupervised clustering or classification problem: Given a set of data points, the software should output an array which assign a label (would be an postive integer) to each data point to indicate the cluster it should belongs to.

3.2 Objectives

In our verification plan, the functional requirements would be our top priority and some of non-functional requirements would also be part of concern.
The fundamental objective will be:

- build confidence in the software correctness. The convergence of the EM training algorithm should be our main concern. We should ensure each update for the training is correct and smooth.
- ensure an accuracy comparable to the pseudo oracle (GMM in scikit-learn package)

While some parts would not be our concern:

- External linear algebra and mathematics library would be used in our software. We assume these libraries we used are verified by their implementation team.

- Basic usability would be ensured but not our main focus, since our library only involves basic function or class call.

3.3 Relevant Documentation

Reader should refer to the SRS document to review how the algorithm works and the software requirement. More implementation would be stated in our other document like MG or MIS.

4 Plan

This section introduces the overall plan for each software development stage. This includes: SRS stage, design stage, verification plan , implementation plan. We also includes tools and pseudo oracle for verification and validation.

4.1 Verification and Validation Team

GMM-EM is reviewed under the followings reviewers. They provide suggestions in different development stages based on the documents and code.

	Role	Description
Prof. Smith	Professor	Review all the documents and code submitted
Xinyu	Primary reviewer	Review and provide suggestion as a domain expert
Valerie	Secondary reviewer	Review SRS document
Seyed Ali	Secondary reviewer	Review VnV document
Hossain	Secondary reviewer	Review MG document

Table 1: Reviewers Team

4.2 SRS Verification Plan

SRS should be verified based on the items listed below:

- Abstract but clear description on goal statement and solution characteristics

- A detailed general system description is given
- IMs and (possibly) TMs and GMs are referenced as appropriate by the requirements. It is a sign that the IMs are not set correctly if there is one or more IMs that are not referenced by any of the requirements.
- All requirements are validatable and abstract.
- Requirements are traceable to where the required details are found in

Our verification and validation team will check for the items above and review the SRS document with GitHub issues. One GitHub issue corresponds to one of the concern or question to our SRS document. Also, team member could create an issue on the overall document with remarks in PDF format.

4.3 Design Verification Plan

The module design could be verified as follows:

- One module corresponds to one issue
- Detailed mathematical formulation and implementation should be liseted here.
- One module corresponds to one issue

Checklist for Module Decomposition.

- One module one secret (unless an explicit exception is made, with a good reason) - all "and"s should be checked.
- The uses relation is a hierarchy.
- Secrets are nouns (generally).
- Traceability matrix between modules and requirements shows every requirement is satisfied by at least on module
- Traceability matrix between modules and requirements shows that every module is used to satisfy at least one requirement

- Traceability matrix between likely changes and modules shows a one to one mapping, or, if this is not the case, explains the exceptions to this rule.
- Level 1 of the decomposition by secrets shows: Hardware-Hiding, Behaviour-Hiding and Software Decision Hiding.
- Behaviour-Hiding modules are related to the requirements
- Software-Decision hiding modules are concepts that need to be introduced, but are not detailed in the requirements
- Each Software Decision Hiding module is used by at least one Behaviour-Hiding Module (if this isn't the case, an explanation should be provided)
- Uses relation is not confused with a data flow chart
- Anticipated changes are a superset of the likely changes in the SRS

The MG quality should fulfill:

- Follow template
- Low coupling
- Satisfies information hiding

4.4 Verification and Validation Plan Verification Plan

Peer reviewers should be verified the plan based on following checkpoints.

- Specific input and Explicit output are stated in test case
- Automated testing and verification tools (included but not limited to linter, profiler, code coverage tool, unit test framework, etc.) are specified for C++.
- Error metric in software testing is specific
- Coverage of the software is enough

- Plans for what to do with description data (performance, usability, etc). This may involve saying what plots will be generated.
- Plans to quantify error for scalar values using relative error
- Plans to quantify error for vector and matrix values using a norm of an error vector (matrix)
- Plans are feasible (can be accomplished with resources available) and ambitious enough.
- Plans for task based inspection, if appropriate
- Traceability between test cases and requirements is summarized (likely in a table)

4.5 Implementation Verification Plan

The testing detail could be referred to Section 5 (System Test) and Section 6 (Unit Test). The static verification for the software would be helped by the static code analyser cppcheck. Further code review could be carried by any other collaborator in this GitHub repository such as professor, primary reviewer and secondary reviewers.

4.6 Automated Testing and Verification Tools

- We will use GoogleTest which supports C++ as our Unit testing framework.
- CI/CD pipeline will be automatic built up by Github Action. Every push or pull command to GitHub will trigger the workflow. The GitHub Action will try to build the software in multi-platform (Windows, Linux , MacOS) with docker container. The unit testing included will be carried out and ensure each update passes the testing. Developers do not need to manually setup for the continuous testing.
- cppcheck which could be used to be linter and a static code analyser for C++.
- Valgrind will be our profiler to monitor the program performance and also check any memory leak exists.

- CMake as a open-source and cross-platform tool will be used to facilitate the build, test and package the software.
- Gcov and LCOV will be the tools to check for the code coverage.

4.7 Software Validation Plan

The software validation is out of our scope.
A pseudo Oracle will be used to validate the correctness of our software GMM-EM. The potential pseudo oracle is the Gaussian mixture model implemented by scikit-learn (sklearn)Pedregosa et al. (2011). We will first call the class from sklearn mixture GaussianMixture and fit the model using fit function in sklearn. Several items will be compared to this function including the parameters in the training model (the class get params function in sklearn will be used) and most importantly the predicted result (the class predict function in sklearn will be used). This would be compare to ensure the correctness and accuracy is at a similar level as the pseudo oracle.

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 Testing for functional requirement R4

To test the correctness of GMM-GM, several datasets will be given to our software. The datasets used are provided and available on our GitHub repository under the dataset folder. The dataset for the first test case is generated by the Python package scipy.stats with multivariate_normal function. The dataset for the second test case is provided by scikit-learn Toy datasets from load_iris. We convert the dataset format from python numpy array to csv file. The output should be an array containing the label (non-negative integer) for the data points. The size of label array should be the same as number of datapoints in the dataset. The first test case (test-id1-two-gaussian) will test the implementation of our algorithm in an ideal case. The second test case will test for a practical case which is common in Data Science.

Our ideal output should correctly indicate each datapoint to the Gaussian distribution it should belongs to. Since Gaussian mixture model fundamentally assumes the datapoint are from the Gaussian distribution, the software should able to capture the features of datapoint and label it to the correct distribution.

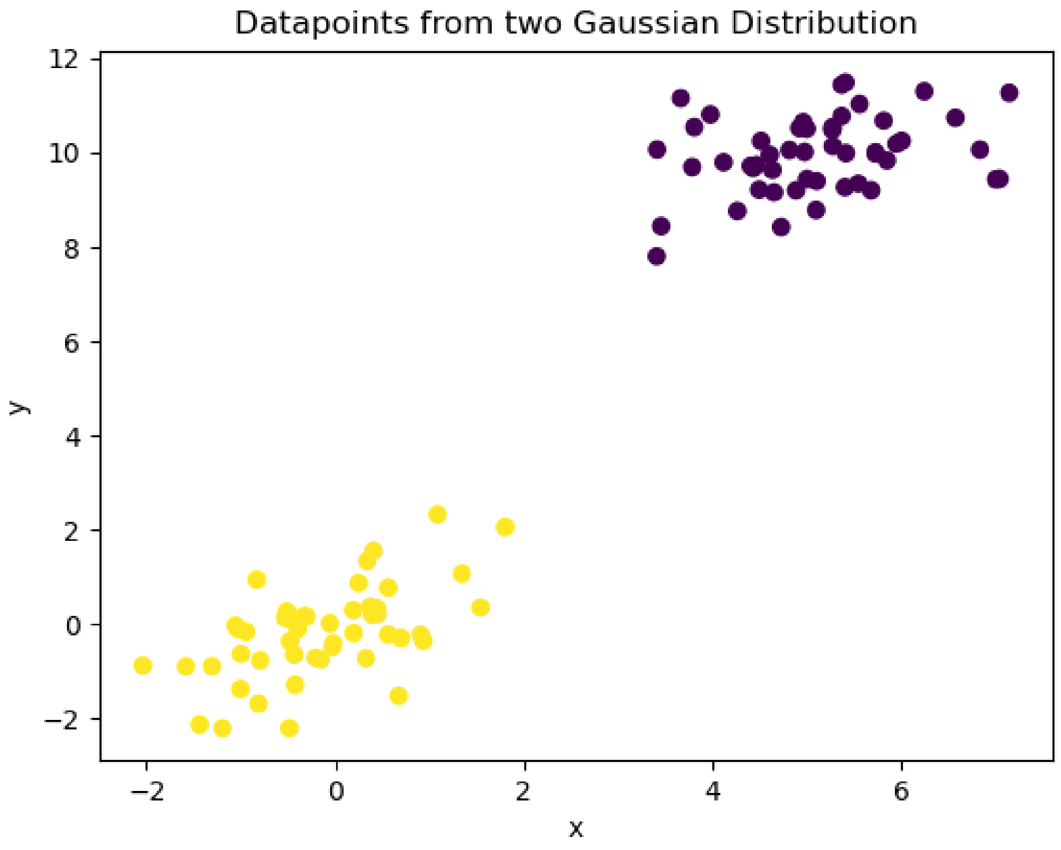

Figure 1: dataset for test-id1-two-gaussian

Test case from Hypothetical situation

1. test-id1-two-gaussian

   The dataset for first test case contains 100 data points, Half (50 datapoints) are generated from a 2D Gaussian distribution with mean =

$[0, 0]$ and covariance $= \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$, Half (50 datapoints) are generated from a 2D Gaussian distribution with mean $= [5, 10]$ and covariance $= \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$. The Figure 1 shows the datapoints in a 2D plane.

Control: Automatic

Initial State: The GMM is called.

Input: An csv file named "two_gaussian.csv" which refer to the dataset containing 100 datapoints. The description could refer to the above.

Output: An 1D array with M elements where the elements $\in \{1, 2, \}$

Test Case Derivation: The output should be number of $\{1, 2\}$ since there are only two Gaussian distribution, and only integer value will be valid since the number of distribution is a discrete value.

How test will be performed: Specific input to software and the software would train the model based on the designed algorithm and we assert our result array to be $\in \{1, 2\}$, otherwise error will throw and terminate the test.

Test case from Real Case The second test case will be a real dataset called iris dataset Fisher (1988) which is provided by scikit-learn toy dataset. The dataset contains 150 datapoints with 4 predictor variables. As we stated, we convert the numpy array from load_iris function and store into csv file name "iris.csv". The dataset is intended to used in classification or clustering. The dataset consists of 3 different types of iris (labels) which is stored in an array of size 150×4. The label array will be stored in another array of size 150×1 which will not be the input of our software but used in the verification stage for accuracy.

2. test-id2-iris

   Control: Automatic

   Initial State: The GMM is called.

Input: an csv file ”iris.csv” containing 150 datapoints , the number of cluster : 3 (corresponds to the three different lables (types) of iris)

Output: An array with 150 elements which each element either is 1 , 2 or 3.

Test Case Derivation: Only three types of iris in dataset and therefore should not exist any number other than 1 ,2 and 3.

How test will be performed: Same as test-id1.

5.2 Tests for Nonfunctional Requirements

This part will focus on the accuracy and performance. The usability will not be heavily tested for our software.

5.2.1 NFR 1: Accuracy

The accuracy test will be also carried in the test for functional requirement 5.1. In addition to output testing, we also check for the accuracy. The accuracy is measured and compared to the pseudo oracle with metric of miss-classification rate and adjusted rand index (ARI). To desrcibe our error for the model, miss-classification rate and adjusted rand index (ARI) are introduced. These two metric take the range from 0 to 1. For miss-classification rate, 0 means no miss-match between true label and prediction, which means no error found in our model output. For ARI, a number close to 1 equals a better classification. ARI = 1 means no mis-match. The following formula and definition is generated by ChatGPT 3.5 with prompt (”adjusted rand index in latex and missclassification rate in latex ”) Miss-classification rate is defined as: The misclassification rate is calculated as:

$$\text{Misclassification Rate} = \frac{\text{Number of Misclassified Instances}}{\text{Total Number of Instances}}$$

ARI is defined as:

$$ARI = \frac{\text{Observed Agreement} - \text{Expected Agreement}}{\text{Max Possible Agreement} - \text{Expected Agreement}}$$

Where:

$$\text{Observed Agreement} = a + d$$

$$\text{Expected Agreement} = \frac{(a+b)(a+c)+(c+d)(b+d)}{n(n-1)/2}$$

$$\text{Max Possible Agreement} = \frac{a+b}{2} + \frac{a+c}{2}$$

Accuracy compared to the pseudo oracle.

1. test-id1

   Detail refers to Section 5.1

2. test-id2

   Detail refers to Section 5.1

5.2.2 NFR 4: Performance

The performance will be tested for several cases. First we concern about the efficiency of the training process. We will profile the running time for each function in our model. The function call we would like to boost will be the calculation in each update of the EM algorithm. This part would be handled by unit test and details are in Section 6. In this section, we will discuss on how performance are concerned in our system level and the goal for performance testing is to figure out the computing limit of our software,i.e. how fast it could be for some tremendously large dataset.

test-id1

Control: Automatic
Initial State: The GMM is called.
Input: M (M is a arbitrary and random positive number, in some edge cases, we would require M to be enough large to test the calculating power of our software.) data points in different dimensions as our dataset will be generated randomly from different Gaussian distribution with distinct mean and covariance. We also need to specific the number of different Gaussian distribution used in this case.
Output: An array with M elements (M denoted the number of points which is a positive integer)

Test Case Derivation: The output should be an array of positive integer and reason is the same as other test cases.

5.3 Traceability Between Test Cases and 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

6 Unit Test Description

The detail of the unit test could be referred to MIS (which would be finished later). The main focus in unit test should be the implementation for the EM training algorithm. The potential unit tests include:

- Check the parameter in model could be successfully initiated randomly (or by K-mean clustering in further implementation)
- Check the correctness of calculation for log-likelihood function and responsibilities. A real-valued number should be returned in this case.
- Convergence of the algorithm should be guaranteed after passing the unit tests on calculation above

6.1 Unit Testing Scope

The unit testing will focus on functions in every module. We aims at testing every single function for the modules, or at least test the function which is key for implementing the clustering algorithm.

6.2 Tests for Functional Requirements

6.2.1 Testing for functional requirement R1: Input Module

4 units testing for valid input, illegal values handling, file error handling, empty data file exception handling.

1. test-id1-valid-input

   Type: automatic

   Initial State:

   Input: an string "inputTest1.csv" that points to a valid csv file , containing only valid values, i.e. real number (named under test folder)

   Output: The expected result for the given inputs

   Test Case Derivation: Justify the expected value given in the Output field

   How test will be performed: google test framework will wrap the function call for csv_to_data from input module. It returns passed case if the size of the output and each elements in output are the same as expected. The expected output is hard-coded in the unit test call.

2. test-id2-illegal-values

   Type: automatic

   Initial State: input module function call : csv_to_data.

   Input: an string "inputTest2.csv" that points to a csv file with an illegal value (a string "string" in the first row.)

   Output: exception throw, an std::invalid_argument that report the data type cannot be cast into double.

   Test Case Derivation: Justify the expected value given in the Output field

   How test will be performed: google test framework will wrap the function call for csv_to_data from input module. It returns passed case if an correct exception throws.

3. test-id3-InvalidCSV

   Type: automatic

   Initial State: input module function call : csv_to_data.

   Input: an string "xxx.csv" which points to an non-exist file.

   Output: exception throw, an std::runtime_error shows File not Found

   Test Case Derivation: Justify the expected value given in the Output field

   How test will be performed: google test framework will wrap the function call for csv_to_data from input module. It returns passed case if an correct exception throws.

4. test-id4-EmptyCSV

   Type: automatic

   Initial State: input module function call : csv_to_data.

   Input: an string "empty.csv" which points to an csv without datapoints.

   Output: exception throw, an invalid argument shows dataset is empty

   Test Case Derivation: Justify the expected value given in the Output field

   How test will be performed: google test framework will wrap the function call for csv_to_data from input module. It returns passed case if an correct exception throws.

6.2.2 Testing for functional requirement R2: Model Initiation Module

2 unit testing for calculating the log-likelihood function and multivariate normal (Gaussian) distribution density function are derived here. We test with hard-coded input and hard-coded expected output. The expected output is using the function in python package (scipy-stats).

5. test-id5-initiate

   Type: automatic

   Initial State: GMM model is called (initiate the number of cluster $K = 4$, where $\leq$ number of datapoints)

   Input: an array of $[1, 2, 3, 4]$ as the input data for GMM model

   Output: std::runtime_error stated that the initiate failed due to number of cluster $K \leq$ number of datapoints.

   Test Case Derivation:

   How test will be performed: google test framework will wrap the function call for model fit (where initiate function is wrapped inside.) from initiation module. It returns passed case if an correct exception throws.

6.2.3 Testing for functional requirement R2: Model Training Module

2 unit testing for calculating the log-likelihood function and multivariate normal (Gaussian) distribution density function are derived here. We test with hard-coded input and hard-coded expected output. The expected output is using the function in python package (scipy-stats).

6. test-id6-likelihood

   Type: automatic

   Initial State: GMM model is initiated with data using model fit function. Followed with function call in likelihood calculation.

   Input: an array of $[1, 2, 3, 4, 5]$ as the input data for GMM model

   Output: a double with value -8.827560617423226 (calculated by external function from python package: scipy-stats)

   Test Case Derivation: We test the actual output with values calculated by external function from python package: scipy-stats. We check for each element in the actual output with expected output. We allow an error with 1e-2 for every element, due to the rounding error.

   How test will be performed: google test framework will wrap the function call for likelihood calculation from training module. It returns passed case if an correct exception throws.

7. test-id7-normal-cal

   Type: automatic

   Initial State: GMM model is initiated with input data using model fit function. Followed with function call in calculating normal distribution pdf.

   Input: an array of $[1, 2, 3]$ as the input data for GMM model

   Output: an array of $[0.23079948, 0.48860251, 0.23079948]$ (calculated by external function from python package: scipy-stats)

   Test Case Derivation: We test the actual output with values calculated by external function from python package: scipy-stats. We check for each element in the actual output with expected output. We allow an error with 1e-2 for every element, due to the rounding error.

   How test will be performed: google test framework will wrap the function call for normal pdf calculation from training module. It returns passed case if an correct exception throws.

6.2.4 Testing for functional requirement R3:

We implemented this as a system for verifying the output with the existing library. Details on Section 5.

6.2.5 Testing for functional requirement R4: Model Predict Module

8. test-id8-no-fit-error

   Type: automatic

   Initial State: GMM model is initiated without fitting

   Input: no input is needed

   Output: Exception throw should appear. An runtime error stated that the model is not fitted.

Test Case Derivation: This test is used to prevent user using the model inappropriately. Our prediction module should always return in positive integer array if it is fitted. If model fit function has not been called, an error should appear to prevent further step.

How test will be performed: google test framework will wrap the function call predict function from model predict module. It returns passed case if an correct exception throws.

9. test-id9-postive-int-unique

   Type: automatic

   Initial State: GMM model is fitted with data

   Input: an array $[1, 2, 5, 6]$, an cluster K = 2 (where is smaller than number of dataset)

   Output: An label array l with size of 4. The non-repeating elements of l should equal to the set $\{1, 2, 3\}$

   Test Case Derivation: This test ensures the prediction matches the requirement from the user. The label arry l shows success for classifying the data into the number of cluster where user requires.

   How test will be performed: google test framework will wrap the function call for predict function from model predict module. It returns passed case if an correct exception throws.

6.2.6 Testing for Output Module

10. test-id10-ValidLabels
    Type: automatic

    Initial State: GMM model is fitted with data

    Input: an array $[1, 2, 5, 6]$, an cluster K = 2 (where is smaller than number of dataset)

    Output: An label array l with size of 4. The non-repeating elements of l should equal to the set $\{1, 2, 3\}$

    Test Case Derivation: This test ensures the prediction matches the requirement from the user. The label arry l shows success for classifying the data into the number of cluster where user requires.

How test will be performed: google test framework will wrap the function call for predict function from model predict module. It returns passed case if an correct exception throws.

11. test-id11-InvalidPath
    Type: automatic

    Initial State: GMM model is fitted with data

    Input: an array $[1, 2, 5, 6]$, an cluster K = 2 (where is smaller than number of dataset)

    Output: An label array l with size of 4. The non-repeating elements of l should equal to the set $\{1, 2, 3\}$

    Test Case Derivation: This test ensures the prediction matches the requirement from the user. The label arry l shows success for classifying the data into the number of cluster where user requires.

    How test will be performed: google test framework will wrap the function call for predict function from model predict module. It returns passed case if an correct exception throws.

6.3 Tests for Nonfunctional Requirements

The testing for nonfunctional requirements focuses on accuracy , portability and performance. Usability and understand-ability is out of the scope of our unit testing. However, these are not suitable for unit testing.

1. Accuracy: It is better to carry out a system test which involves every modules in our GMM-EM. Since the result is produced only when the software runs every module.

2. Performance: The same reason applies. We do not aim at using unit test for each function in profiling. Instead, we profile for every function call when running the entire software GMM-EM, since entire program take less than few seconds.

3. Portability: It will test through CI/CD Automated testing. When any push or pull request occurs, the GitHub Action will try to build the software in a wrapped docker container where implement different Operating system.

6.4 Traceability Between Test Cases and Modules

Reader could refer to the heading for every section. The traceable matrices here simplify the result:

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	UT 9		
M6 (Output module)	UT 10	UT 11		

Table 3: Traceability Between Test Cases and Modules

References

R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.