Project Title: System Verification and Validation Plan for GMM-EM

WONG, Kim Ying

February 20, 2024

Revision History

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

Contents

List of Tables

Remove this section if it isn't needed

List of Figures

Remove this section if it isn't needed

1 Symbols, Abbreviations, and Acronyms

symbol	**description**
T	Test

symbols, abbreviations, or acronyms — you can simply reference the SRS (?) tables, if appropriate

Remove this section if it isn't needed

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

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

4.3 Design Verification Plan

The module design could be verified as follows:

- One module correspond to one issue and detailed mathematical formulation and implementation should be liseted here.

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 a specific programming language
- Error metric in software testing is specific

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. 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.

- The portability to different operating system will be ensured by Docker.
- 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

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, a certain dataset will be given to our software. The output should be an array containing the label (non-negative integer) for the data points. The first test case (test-id1) will test the implementation of our algorithm. We generate three sets of data points from 1D Gaussian distribution with different mean and covariance as our input. 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. The Figure 1 shows three Gaussian distribution in a 2D plane. We here shows an example. Denote red, green and blue distribution to be $\{1, 2, 3\}$. The software should label 1 to the a datapoint from the red distribution. In our testing, we need to ensure the output will always belongs to the set $\{1, 2, 3\}$.

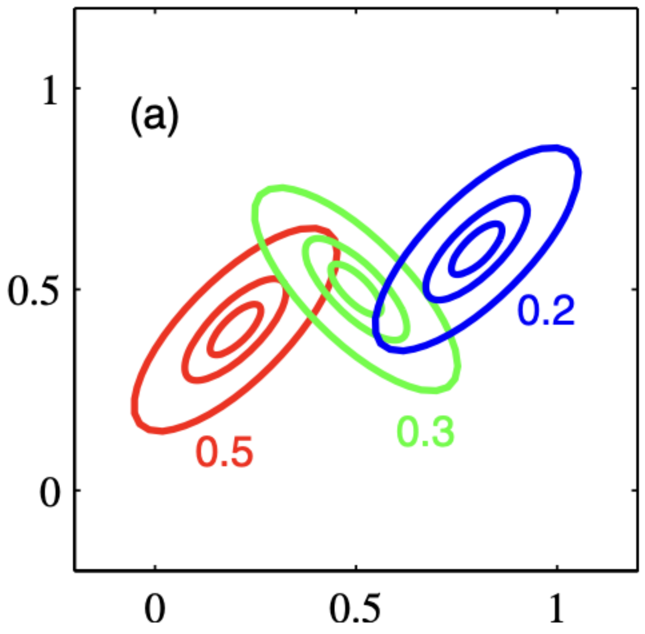

Figure 1: A Sample Testing input Bishop (2006)

Test case from Hypothetical situation

1. test-id1

   Control: Automatic

   Initial State: The GMM is called.

   Input: 1D array with M data points generated from a three 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 1D array with M elements where the elements $\in \{1, 2, 3\}$

   Test Case Derivation: The output should be number of $\{1, 2, 3\}$ since there are only three 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 $\in \{1, 2, 3\}$, otherwise error will throw and terminate the test.

Test case from Real Case The second test case will be a real dataset from Data Science called Iris dataset. The dataset is intendted to used in classification or clustering. The dataset consists of 3 different types of iris 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

   Control: Automatic

   Initial State: The GMM is called.

   Input: Iris dataset: An array of $R^{150\times4}$ and specific the number of types of iris (which is 3 here.)

   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 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). We aim at any of the metric to be within error of $\epsilon \leq 10\%$. The confusion matrix could be constructed and these metric could be calculated from the confusion matrix. 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:

$$MisclassificationRate = \frac{NumberofMisclassifiedInstances}{TotalNumberofInstances}$$

ARI is defined as:

$$ARI = \frac{ObservedAgreement - ExpectedAgreement}{MaxPossibleAgreement - ExpectedAgreement}$$

Where:

$$ObservedAgreement = a + d$$

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

$$MaxPossibleAgreement = \frac{a+b}{2} + \frac{a+c}{2}$$

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 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
5.1.1		X
5.2.1	X	
5.2.2	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

Details will be filled later after MG and MIS document.

6.1 Unit Testing Scope

Details will be filled later after MG and MIS document.

6.2 Tests for Functional Requirements

Details will be filled later after MG and MIS document.

6.2.1 Module 1

Details will be filled later after MG and MIS document.

1. test-id1

   Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

   Initial State:

   Input:

   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:

2. test-id2

   Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

   Initial State:

   Input:

   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:

3. ...

6.3 Tests for Nonfunctional Requirements

Details will be filled later after MG and MIS document.

6.3.1 Module 2

1. test-id1

   Type: Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

Details will be filled later after MG and MIS document.

References

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

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.