Polytechnique Montréal

LOG8371E

Software Quality Engineering

Assignment 1 - Quality plan and tests

Students :
Yuri Azar - 1780762
Jérôme Désilets - 1684912

Teaching Assistant:
Mahsa Hadian

Friday October 4th 2019

Contents

1 Introduction

The complexity of today's softwares is constantly increasing. With it, the need for quality assurance is making itself more and more essential, especially for larger projects. This is why this report aims to define a software quality plan for Weka, a software created by the University of Waikato in New Zealand for machine learning algorithms for data mining tasks.

More specifically, this report contains 5 sections. First, 5 features which have been chosen are described and then analyzed according to 3 quality criteria: Functionality, Reliability, Maintainability. Following these two sections, a collection of tests will be explained, preliminary results will be explained, and a schedule will be made. Next, a continuous integration plan which is using these tests will be done in Travis CI and then demonstrated. Following that, we will explain the consequences of the addition of a new algorithm and the changes that this addition makes to the previously discussed parts.

2 Team organization

A team constituted of 5 people working in software development has been assembled. The member's names are: Yuri, Jérôme, Bobby, Dandelion, Percival.

Jérôme is the developer and will work on unit and functional tests. Bobby is the architect and can help Jerome write some unit or functional tests, Dandelion is the Business Analyst and Percival is the Product Owner, who will write the acceptance tests along with the rest of the team, but mainly the Business Analyst, who is Dandelion. Finally, Yuri will be the project manager, so he will not do any test and instead will make sure that the development follows the schedule.

3 Features

Before we mention the quality criteria and their application to the software, we have to explain which features are highlighted by this current plan so that we can better understand how to evaluate them.

Before they are described, it is important to realize that the features that we have chosen are part of two main categories: classification and clusterer algorithms. Classification algorithms categorize data into a given number of already existing classes, whereas clusterer algorithms involve grouping data points into specific groups with respect to their similarities.

As you may realize, these algorithms are statistically sophisticated. It is important that the quality of their implementation into the software is at its maximum so that the data scientists using them are confident that their use will not obstruct their work.

3.1 Naive Bayes

The Naive Bayes algorithm is a classifier algorithm with an assumption of independance among predictors. It assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature [1]

3.2 Multinomial naive Bayes

The Multinomial Naive Bayes Algorithm is an extension of the Naive Bayes algorithm, which assumes that each of the feature uses a multinomial distribution. [2]

3.3 Expectation-Maximization

The Expectation-Maximization algorithm is a clustering algorithm. More specifically, it is an iterative method to find maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. em

3.4 Farthest-first (clusterer)

The Farthest First algorithm is a clustering algorithm mostly used in computational geometry. The farthest-first traversal of a bounded metric space is a sequence of points in the space, where the first point is selected arbitrarily and each successive point is as far as possible from the set of previously-selected points. [4]

3.5 Cobweb (clusterer)

The Cobweb algorithm is a classifier algorithm with an assumption that incrementally organizes observations into a classification tree. Each node in a classification tree represents a class (concept) and is labeled by a probabilistic concept that summarizes the attribute-value distributions of objects classified under the node. This classification tree can be used to predict missing attributes or the class of a new object. [5]

4 Quality criteria

Each of the previously described features will have their quality validated according to the following sub-criteria, objectives and evaluation methods. All of the following validation metrics and methods will be applied valid for every feature. We have chosen to evaluate two sub-criteria per quality criteria, because they are considered important enough to convey the level with which the criteria is covered. Each of these two sub-criteria are considered as the most important to evaluate for the current project in their respective criteria.

4.1 Functionality Criteria

The functionality criteria evaluates the degree to which a product is able to meet the needs according to its specified requirements.

Two sub-criteria will be evaluated. There is Functional completeness, which is the degree to which the features are complete according to the requirements. In this particular instance, this sub-criteria should be met because specific already implemented features have been chosen. There is also Functional correctness, which is the ability for a feature to provide the correct results. In our case, the output of each algorithm is expected to be right.

The following table describes the objectives, validation metrics and validation meth-ods for each sub-criteria, which will be used to judge whether or not the reliabilitycriteria is met

Quality Objectives for Functionality Criteria			
Sub-Criteria	**Objectives**	**Validation Metrics**	**Validation Methods**
Functional completeness	More than 95% of the functional requirements are implemented	Number of implemented requirements according to requirements specifications	Unit Tests and Functional Tests
Functional correctness	Feature provides expected results more than 95% of the time	Number of times the expected result is output	Unit Tests and Functional Tests

4.2 Reliability Criteria

The reliability criteria evaluates the degree to which a system can perform under specified conditions for a specified period of time.

Two sub-criteria will be evaluated for this specific criteria: Maturity, which is the degree to which a system's evolution is mature and stable enough for it to be considered reliable, and Availability, which describes the degree to which a feature is accessible for use.

The following table describes the objectives, validation metrics and validation methods for each sub-criteria, which will be used to judge whether or not the reliability criteria is met.

Quality Objectives for Reliability Criteria			
Sub-Criteria	**Objectives**	**Validation Metrics**	**Validation Methods**
Maturity	Software Maturity Index value should be greater than 0.85	See annex A for maturity formula	Calculation
Availability	Result of availability formula is equal to or greater than 99.9%	See annex A for availability formula	Calculation

4.3 Maintainability Criteria

The maintainability criteria evaluates the degree of ease with which the system can be modified, improved, adapted.

Two sub-criteria will be evaluated for this specific criteria: Modifiability, which is the degree to which a feature can be easily modified without affecting the rest of its parts, and Testability, which is the degree of ease to which tests can be evaluated.

The following table describes the objectives, validation metrics and validation methods for each sub-criteria, which will be used to judge whether or not the maintainability criteria is met.

Quality Objectives for Maintainability Criteria			
Sub-Criteria	**Objectives**	**Validation Metrics**	**Validation Methods**
Modifiability	4 stars or higher SIG rating	**Duplication:** At most 4.6% of features are redundant lines of code. **Unit complexity:** a Mccabe of at most 5, at least 74.8%, a McCabe above 5 of at most 25.2%, a McCabe above 10 of at most 10.0%, a McCabe above 25 of at most 1.5% **Module coupling:** Fan-in of 51+ modules of at most 6.6%, Fan-in of 21-50 modules of at most 13.8%, Fan-in of 11-20 modules of at most 21.6%, Fan in of 1-10 modules without any constraint	Code analysis to extract desired and pertinent statistics such as : - number of branches in a method - number of redundant lines of code
Testability	4 stars or higher SIG rating	**Unit complexity:** a Mccabe of at most 5, at least 74.8%, a McCabe above 5 of at most 25.2%, a McCabe above 10 of at most 10.0%, a McCabe above 25 of at most 1.5%	Code analysis to extract desired and pertinent statistics such as : - number of branches in a method - number of redundant lines of code

5 Tests

In this current section, we will plan the tests that need to be made in order to validate the mentioned quality criteria. Since it is only a plan, we cannot yet say whether or not our metrics are fulfilled for each criteria. However, we can establish a preliminary analysis and then establish which members of our team will be responsible to realize the required tests and analysis.

First off, we will whether each criteria is already validated or needs to be, so that we can create a compendium of our tests and their descriptions and then schedule their realization by our team.

5.1 Preliminary Sub-Criteria Validity

Functional completeness

Since we do not have the functional requirements that were planned by the development team, we cannot hereby address whether or not this criteria is successfully met. Therefore, our team will have to contact the Weka development team to validate this criteria. However, with unit and functional tests we can verify that the requirements as understood by the unit testers are met, since all the tests are working for all the features.

Functional correctness

With the already provided unit tests, we can verify that the output for an algorithm's test is same as the expected ones, which is the case according to our results, which are presented in a further section. Therefore, the only thing that has to be done is run the tests.

Maturity

Since the calculations require a great amount of data for this sub-criteria, the collection and the computing will be performed by members of our team in a future sprint so that we can be certain of the maturity.

Availability

Preliminar usage of the Weka features shows that the software is victim to few crashes, which would let us know that the availability sub-criteria seems to be met. However, more data needs to be collected and computed so that we can scientifically determine the software's availability percentage.

Modifiability

In order to validate the modifiability sub-criteria , we need to extract the percentage of redundant lines of code and the number of branches in every method. In order to achieve that, we will examine and analyze the code.

Testability

In order to validate the testability sub-criteria, we need to examine the number of branches in every method. However, the huge number of unit tests available can be used to test the Testability of the weka software.

Because we need to wait for the experts' results, we can't validate the modifiability nor the testability sub-criteria.

5.2 Tests description

Sub-Criteria	**Scope**	**Test process**	**Action in case of failure**
Functional completeness	Individual feature's class	Run the feature's test suite, gather evidence for the stakeholder's need	Implement the missing functionnalities
Functional correctness	Individual feature's class	Run the feature's testUpdatingEquality() test	Debug the class
Maturity	Software	Gather the data necessary to compute the maturity formula, compute the maturity formula	Keep developing the software until criteria is met
Availability	Software	Gather the data necessary to compute the availability formula, compute the availability formula	Fix bugs causing crashes according to the collected data until criteria is met
Modifiability	Software	Analyze duplication, unit complexity, module coupling	Refactor the code until criteria are met
Testability	Software	Analyze unit complexity	Refactor the code until criterias are met

5.3 Testing schedule

The schedule to establish our software's qualities is quite simple and short. Since the unit tests are already provided, most of the job is data collection. Therefore, a sprint of 1 week should be enough for our team to accomplish the testing plan. We have allowed time for the team to fix the issues that may be encountered, but the sprint after this planned sprint could also be entirely dedicated to this since it may be a huge task.

Monday, October 14th (Day 1):

Morning:

- Dandelion will work with Percival in obtaining the system's requirements.
- Jérôme will work with Bobby in running all the tests and verifying their results. It should not be that long since all the tests are already made. This will validate the functional correctness sub-criteria for all the features.

Afternoon:

- Dandelion will work with Percival in determining whether the tests imply Functional completeness for all the features.
- Jérôme and Bobby will do a code analysis to extract the data surrounding the modifiability and testability sub-criteria and compute the formula. This will validate the modifiability and the testability sub-criteria.

Tuesday, October 15th (Day 2):

Morning:

- Dandelion will work with Percival in determining the system's requirements.
- Jérôme will work with Bobby in running all the tests and verifying their results. It should not be that long since all the tests are already made - This will validate the Functional correctness sub-criteria for all the features.

  Afternoon:

- Jérôme and Bobby will gather the necessary data and compute the maturity formula. This will help validate the maturity criteria for all the features.

Wednesday, October 16th (Day 3):

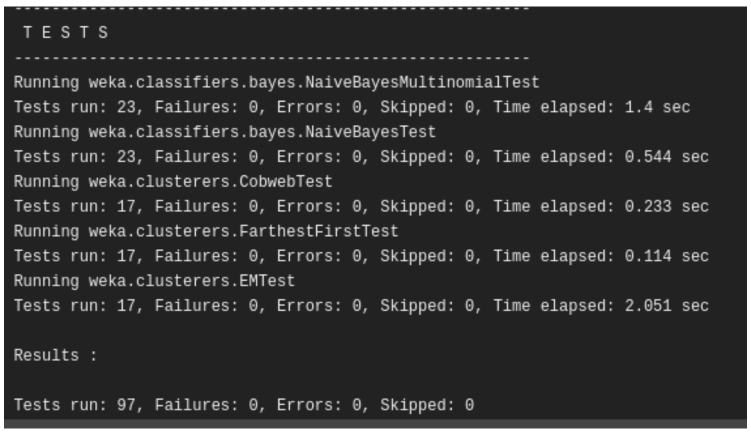
Morning:

- Jérôme and Bobby will gather the necessary data and compute the availability formula. This will help validate the availability criteria for all the features.

Thursday, October 17th (Day 4) to Friday, October 16th (Day 5): Any issues that were encountered in the previous days are to be fixed by Jérôme and Bobby so as to increase the software's quality criteria's metrics to their apex.

5.4 Test reports

We have been able to use pre-designed test cases by the Weka team to run our unit tests. Here is an example test report that is created using our continuous integration pipeline in Travis CI, which we will detail in the next section.



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running weka.classifiers.bayes.NaiveBayesMultinomialTest
Tests run: 23, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.4 sec
Running weka.classifiers.bayes.NaiveBayesTest
Tests run: 23, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.544 sec
Running weka.clusterers.CobwebTest
Tests run: 17, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.233 sec
Running weka.clusterers.FarthestFirstTest
Tests run: 17, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.114 sec
Running weka.clusterers.EMTest
Tests run: 17, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.051 sec

Results :

Tests run: 97, Failures: 0, Errors: 0, Skipped: 0
```

As you can see, tests are being ran for each of our features and all of them are successful, which means that our criteria that are concerned by unit and functional tests are valid.

6 Continuous integration plan

The continuous integration plan that has been designed for this current project relies on two main components: Github and Travis CI. The developers will use Gitlab so that they can develop the project concurrently, using its many features. For every change or addition that is made, a developer will commit its modifications and then push it onto the Github repository.

Synchronized to the Github repository is a Travis CI implementation, which, for every commit that is made on every branch, builds and executes the tests that have been designed. Every time a new test is pushed, it will also be executed. If the build or a test fails, an error is sent to the developers who then fix the problem. This loop keeps on going until there are no more commits made to the project, which means that the project is done. The continuous integration is managed by developer who configures a file named .travis.yml placed on the main directory of the Github repository. It has already been set up and the results are already being sent to the members of the project.

Since the Github repository and the Travis CI are both set up for the project, the continuous integration for this project is well set up. Only when the project's build configuration needs to be changed will the Travis CI configuration need to be changed.

7 Additional Feature

We have chosen to add the Apriori algorithm, which is an association algorithm. It finds relations between the given variables. Apriori algorithm is a classical algorithm in data mining. It is used for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store. [6]

Now that this feature's nature has been established, since it is a feature similar to the others (an algorithm), there is no need to modify the quality plan. The criteria, sub-criteria, metrics an validation methods will stay the same for this feature. To validate this feature's quality, we have to execute all the validation methods that have been made for all the feature for this new feature, and we also have to add a regression test because we have added a feature on top of the pre-defined. This test is taken care of already by Weka's developers. Therefore, if the test results are still successful once the new feature is added, the addition did not introduce additional defects.

With this addition, we have to decide whether or nout our testing schedule has to be modified, since code analysis, calculations and unit tests have to be made for us to be sure that it meets our quality criteria. However, since it is an addition of one relatively small feature and each sub-criteria is validated in batch, the added delay should not represent much time. Therefore, we do not consider it needed to adapt the testing schedule.

8 Continuous Integration Example

The video located at: `https://youtu.be/f0xlqSfU_nQ` has been made by our team to demonstrate of how our process for continuous integration works.

9 Annex A

9.1 Formula to calculate Software Maturity Index

$$SMI = MT - \frac{F_a + F_c + F_d}{MT}$$

Where :

SMI is the Software Maturity Index

MT is the number of software functions in the current release

F_a is the number of functions that contain additions to the previous release

F_c is the number of functions that contain changes from the previous release

F_d is the number of functions that are deleted from the previous release

9.2 Formula to calculate Availability

$$Availability = (\frac{MTBF}{MTBF + MTTR}) * 100$$

Where :

$Availability$ is the Software Availability

$MTBF$ is the Mean Time Between Failure

$MTTR$ is the Mean Time To Recovery

References

[1] 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R, Retrieved October 12 2019(n.d.), from
`https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained`

[2] Difference between naive Bayes & multinomial naive Bayes(n.d.), Retrieved October 12 2019, from
`https://stats.stackexchange.com/questions/33185/difference-between-naive-bayes-mul`

[3] Expectation-Maximization algorithm(n.d.), Retrieved October 12 2019, from
`https://en.wikipedia.org/wiki/Expectation%E2%80%93`
`maximization_algorithm`

[4] Farthest-first travaersal(n.d.), Retrieved October 12 2019, from
`https://en.wikipedia.org/wiki/Farthest-first_traversal`

[5] Cobweb (clustering)(n.d.), Retrieved October 12 2019, from
`https://en.wikipedia.org/wiki/Cobweb_(clustering)`

[6] Apriori algorithm(n.d.), Retrieved October 12 2019, from
`https://en.wikipedia.org/wiki/Apriori_algorithm`