/

**Course : LOG8371E**

Software Quality Engineering

**Assignment #1 : Software Quality plan, testing plan and continuous Integration**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **By :** | | |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | Students Identification | |  |  |
|  |  |  |  |  |  |
|  | Student #1 |  | Freddy Sossa | |  |
|  |  |  |  | |  |
|  | Student #2 |  | Sahil Dahake | |  |
|  |  |  |  |  |  |

**Submitted to :**

Mahsa Hadian

**On :**

October 12, 2019

**Table of contents**

Introduction

1. Software Quality Plan
2. Tests
3. Continuous Integration Plan
4. New Algorithm
5. Demonstrative Video

**Introduction**

Weka (Waikato Environment for Knowledge Analysis) is an open source software developed at the University of Waikato in New Zealand. It’s a data mining application that contains a collection of tools and algorithms for data analysis and predictive modeling. This is app is commonly use in agricultural domains and more recently for educational and research purposes. This tool is being also used for academic and research purposes, so it is important to ensure that the results of the data processing done using it are accurate and that the application is working properly. It will therefore be useful to perform a series of tests to ensure the quality and smooth operation of the software. This document serves as a software quality plan. We will be working on two classifications and clustering algorithms and one regression algorithm.

Researchers use this software and rely on it to advance their research. Whether they are researchers in artificial intelligence or researchers in other fields using artificial intelligence during their search, it is important for them that the software has a good quality. It is crucial to properly process the data to obtain relevant results for researchers.

1. **Software Quality Plan**

This document will essentially cover five main features we have chosen. We can therefore say that all those features are considered as our functional requirements. First of all, it’s important to describe those features in order to have an idea of which tasks each feature perform.

1. Main features

* Classifications algorithms
* NaiveBayes Algorithm

Naive Bayes Algorithm is a technique that helps to construct classifiers. Classifiers are the models that classify the problem instances and give them class labels which are represented as vectors of predictors or feature values. It is based on the Bayes Theorem. It is called naive Bayes because it assumes that the value of a feature is independent of the other feature i.e. changing the value of a feature would not affect the value of the other feature. It is also called as idiot Bayes due to the same reason. This algorithm works efficiently for large data sets, hence best suited for real-time predictions.[1]

* LinearRegression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering, and the number of independent variables being used. [2]

* Clustering Algorithms
* FarthestFirst

In [computational geometry](https://en.wikipedia.org/wiki/Computational_geometry), the farthest-first traversal of a bounded [metric space](https://en.wikipedia.org/wiki/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. The same concept can also be applied to a [finite set](https://en.wikipedia.org/wiki/Finite_set) of geometric points, by restricting the selected points to belong to the set or equivalently by considering the finite metric space generated by these points. For a finite metric space or finite set of geometric points, the resulting sequence forms a [permutation](https://en.wikipedia.org/wiki/Permutation) of the points, known as the greedy permutation. [3]

* HierachicalCluster

Hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis) which seeks to build a [hierarchy](https://en.wikipedia.org/wiki/Hierarchy) of clusters. Strategies for hierarchical clustering generally fall into two types:[[1]](https://en.wikipedia.org/wiki/Hierarchical_clustering#cite_note-clusteringMethods-1)

* Agglomerative: This is a "[bottom-up](https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design)" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
* Divisive: This is a "[top-down](https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design)" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined in a [greedy](https://en.wikipedia.org/wiki/Greedy_algorithm) manner. The results of hierarchical clustering are usually presented in a [dendrogram](https://en.wikipedia.org/wiki/Dendrogram). [4]

* Association algorithms
* FPGrowth

FpGrowth Algorithm (Frequent pattern growth) is an improvement of apriori algorithm. FP growth algorithm used for finding frequent itemset in a transaction database without candidate generation. It allows frequent itemset discovery without candidate itemset generation. Two step approaches as building a compact data structure called the FP-tree (built using 2 passes over the data-set) and extracting frequent itemsets directly from the FP-tree (traversal through FP-Tree).

1. Criterias, Objectives and Validation measures

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Sub-Criteria** | **Validation measures** |
| Functionality | Functional correctness | The system delivers correct results 99% of the time |
| Functional completeness | The reviewer of the patch judges the functionality added relevant. |
| Reliability | Recoverability | The system must be able to recover anytime that a crash occurs |
| Fault tolerance | The system must correct itself in case of errors so as not to mislead the researchers 90% of the time. |
| Maintainability | Analysability | Adding a feature changes to maximum 30 lines of existing code. |
| Reusability | Cannot be tested automatically. |
| Testability | 70% of the functional requirements or not  functional are covered by the tests. |

1. Validation Strategy

In our project, the validation will be a set of many activities. First of all, we will a combination of **unit tests and functional tests**. Our testing strategy will be incremental. Units test are the first tool to make sure that the written code is working and is getting the job done. Making sure the code is working properly is the first step. We will also be using **reviews** as validation strategy. We are not the final users, so it’s important for the final user to have their say on the work. The thing is that it’s important to make sure that they understand easily how it works and is working the way they expected it to work. It’s also useful to early detect bugs and problems.

1. Planification

* Team and work

We are working in a team of 4 people for the tests : 2 Developers, 1 Functional Analysts and 1 Tester. The team will be working in a Scrum agile method. The project will be on two months, then eight weeks. Each sprint will last three weeks except the last one which will last 2 weeks. We have chosen a duration of 3 weeks because it’s the optimal duration ( 4 is enough and 2 is not enough to finish what is started).

1. **Tests**
2. Tests Plan
   1. Introduction

To ensure that we have a quality product, it is imperative to test our software and validate the results. A test plan is needed to properly target relevant tests for our software. This test plan will include all of the requirements to be tested as well as our testing strategies with the different types of tests that we will use.

* 1. Requirements

Now, we have to identify the requirements that will be tested (functional and non-functional). Our functional requirements are basically the functionalities of our reduced version of Weka while our non-functional requirements match our quality criteria.

1.2.1. Functional requirements

1.2.1.1. Association algorithm FPGrowth

1.2.1.2. Classification algorithm NaiveBayes

1.2.1.3. Classification algorithm LinearRegression

1.2.1.4. Clustering algorithm FarthestFirst

1.2.1.5. Clustering algorithm HierarchicalCluster

1.2.2. Non-functional requirements

1.2.2.1. Reliability

|  |  |
| --- | --- |
| Recoverability | It involves a set of policies, tools and procedures to enable the recovery or continuation of vital technology infrastructure and systems following a [natural](https://en.wikipedia.org/wiki/Natural_disaster) or [human-induced](https://en.wikipedia.org/wiki/Man-made_hazards) [disaster](https://en.wikipedia.org/wiki/Disaster). It focuses on the IT or [technology systems](https://en.wikipedia.org/wiki/Technology_systems) supporting critical business functions,as opposed to [business continuity](https://en.wikipedia.org/wiki/Business_continuity), which involves keeping all essential aspects of a business functioning despite significant disruptive events. |
| Fault tolerance | Enable a [system](https://en.wikipedia.org/wiki/System) to continue operating properly in the event of the failure of (or one or more faults within) some of its components |

1.2.2.3. Maintainability

|  |  |
| --- | --- |
| Modularity | Degree to which a system or program is composed of components so that a change to a component has an impact minimal on others |
| Reusability | Degree to which an element can be used in more than one system or in construction other elements |
| Testability | Degree of efficiency and effectiveness with which a test criterion can be established for a system, a product or component and presence of tests that can determine if these criteria are met |

1. Tests description

2.1. Unit tests

2.1.1 Association algorithms

The set of unit tests for association algorithms (FPGrowth) is found in the *AbstractAssociatorTest* class. We have:

* testAttributes: tests whether the Associator can handle different types of attributes and if not, if the raised exception is correct.
* testBuildInitialization: tests whether the Associator correctly initializes in the buildAssociator method
* testCanonicalUserOptions: tests whether the user-supplied options stay the same after setting, getting, and re-setting again.
* testClassAsNthAttribute: Checks whether the associator can support the attribute class like the Nth attribute
* testDatasetIntegrity: Tests if the modifier modifies the dataset during training
* testDefaultOptions: Tests if the default settings are processed correctly
* testGlobalInfo: Tests for a globalInfo method
* testInstanceWeights: Tests if the associator handles weight instances correctly
* testListOptions: Tests the list of options
* testMissingClass: Tests whether the associator can handle missing class values
* testMissingPredictors: Tests if the mapper can handle missing predictors
* testNClasses: Tests if the associator supports N classes
* testRegression: Execute a regression test - this verifies that the output of the object tested corresponds to that of a reference version.
* testRemainingOptions: Test if there are any remaining options
* testResettingOptions: Tests the reset of the default options
* testSerialVersionUID: Tests if the schema declares a serialVersionUID
* testSetOptions: Tests the setting of options
* testToolTips: Tests tool tips
* testZeroTraining: Tests if the associator can manage zero training instance

2.1.2. Classification algorithms

The set of unit tests for classification algorithms (LinearRegression and NaiveBayes) is found in the *AbstractClassifierTest* class. We have:

* testAttributes: Tests if the classifier can support different types of attributes and otherwise if the exception is correct
* testBuildInitialization: Tests if the classifier initializes correctly in the buildClassifier method
* testCanonicalUserOptions: Checks whether the user-supplied options remain the same after setup, get and reset
* testClassAsNthAttribute: Checks whether the classifier can support the attribute class like the Nth attribute
* testDatasetIntegrity: Tests if the classifier modified the dataset during training
* testDefaultOptions: Tests if the default settings are processed correctly
* testGlobalInfo: Tests for a globalInfo method
* testInstanceWeights: Tests whether the classifier correctly handles instances of weight
* testListOptions: Tests the list of options
* testMissingClass: Tests if the classifier can handle class values missing
* testMissingPredictors: Tests if the classifier can handle missing predictors
* testNClasses: Tests if the classifier supports N classes
* testOnlyClass: Tests whether the classifier processes data that contains only one class attribute
* testRegression: Execute a regression test - this verifies that the output of the object
* tested corresponds to that of a reference version.
* testRemainingOptions: Test if there are any remaining options
* testResettingOptions: Tests the reset of the default options
* testSerialVersionUID: Tests if the schema declares a serialVersionUID
* testSetOptions: Tests the setting of options
* testToolTips: Tests tool tips
* testToString: Tests whether the toString method of the classifier works even if the classifier has not been built yet
* testUpdatingEquality: Tests whether the classifier produces the same model when it is incrementally or batchwise
* testUseOfTestClassValue: Tests if the classifier mistakenly uses the class value test instances (if provided)
* testZeroTraining: Tests if the classifier can handle zero training instance

2.1.3. Clustering algorithms

The set of unit tests for classification algorithms (FarthestFirst and HierarchicalCluster) is found in the *AbstractClustererTest* class. We have:

* testAttributes: Tests if the clusterer can support different types of attributes and otherwise if the exception is correct
* testBuildInitialization: Tests if the clusterer initializes correctly in the method buildClusterer
* testCanonicalUserOptions: Checks whether the user-supplied options remain the same after setup, get and reset
* testDatasetIntegrity: Tests if the clusterer modifies the dataset during training
* testDefaultOptions: Tests if the default settings are processed correctly
* testGlobalInfo: Tests for a globalInfo method
* testInstanceWeights: Tests if the clusterer handles weight instances correctly
* testListOptions: Tests the list of options
* testMissingPredictors: Tests if the clusterer can handle missing predictors
* testRegression: Execute a regression test - this verifies that the output of the object
* tested corresponds to that of a reference version.
* testRemainingOptions: Test if there are any remaining options
* testResettingOptions: Tests the reset of the default options
* testSerialVersionUID: Tests if the schema declares a serialVersionUID
* testSetOptions: Tests the setting of options
* testToolTips: Tests tool tips
* testUpdatingEquality: Tests if the clusterer produces the same model when it is incrementally or batchwise
* testZeroTraining: Tests if the clusterer can manage zero training instance

2.2. Test suites

All the tests will be done with the libraries *junit.framework.Test* and *junit.framework.TestSuite*. These make it possible to run all the tests of the test classes target are FPGrowthTest, NaiveBayesTest, LinearRegressionTest, FarthestFirstTest, and HierarchicalClustererTest.

2.3. Regression Tests

The regression tests will be done using the automation of the tests on

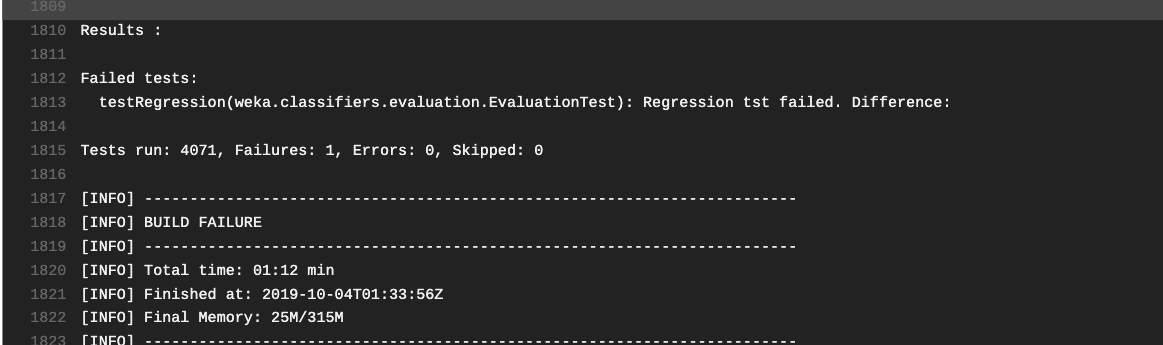
Travis CI. Each modification to the project will launch all the software tests which will quickly detect if the change causes a regression.

2.4. Tests and Criterias

|  |  |
| --- | --- |
| **Sub-Criterias** | **Tests** |
| Functional Correctness | testAttributes, testBuildInitialization, testClassAsNthAttribute, testDefaultOptions |
| Functional Completeness | testNClasses, testMissingPredictors |
| Recoverability | testDataSetIntegrity, testResettingOptions, testBuildInitialization, testDefaultOptions |
| Fault-tolerance | testDataSetIntegrity, testResettingOptions, testDefaultOptions |
| Analysability | Tests coverage |
| Testability | Each unit test contributes to check the testability of the system at this level |
| Modifiability | testRegression, testSetOptions, testAttributes |

1. Testing Report

All the unit tests listed above go for our reduced version of the Weka software. Our tests suites being composed directly of these, we note that the Test suites of our five algorithms pass everything equally. In the special case of our regression tests, we need several builds to be able to confirm the passing the regression test. As we can see in our list of builds on Travis, we have no regression from one build to another. In fact, all of our tests go from one build to another. The main issue is that one of our tests not passing by default. Even without modifying anything, there is one failing test. That means there is probably something broken or not working in the application. That requires more investigation to identify the bug and to fix it in next versions of the app. Anyway, each criteria was tested. Since one test is failing, we can conclude that the functional completeness and the functional correctness criterias are failing.



1. **Continuous Integration Plan**

Ideally, every change made should improves the code, while maintaining the quality of the previous code. But more the code grows, more difficult it becomes to make sure that changes doesn’t bring errors. This is why we introduce the test plans, discussed previously. The integration plan will discuss the methods used to ensure that continuous integration occurs effectively.

* Tools
* GitHub

We started by doing a fork of https://github.com/Waikato/weka-3.8. This

repository contains the latest stable version of Weka. To make our changes we worked on the forked https://github.com/fredsossa1/weka-3.8 This deposit requires sweaters request to integrate it into the master, and any change automatically launches a build on Travis.

* Travis CI

Travis CI is a free software and online service used to compile, test and deploy the source code of the software developed, particularly in connection with the GitHub source code hosting service. The process is pretty simple. First of all, we create a Travis CI account with GitHub, then linked up Travis and GitHub by giving third party authorizations to Travis. Afterwards, we add the weka project to Travis. The main and most important part is to add the *.travil.yml* file to GitHub. This file is the one used by Travis as a configuration file to know how to build and execute tests for the project. Our *.travil.yml* file looks like :



It specify that the project language is java using the jdk8 package. We are working on a trusty Linux distribution. And before executing build and tests, we have to access the weka folder.

1. **New Algorithm**

We decided to add the "KStart" classification algorithm to our Weka app. To maintain the quality level of our system, it is essential the new algorithm is functional, bug-free and can coexist with the existing algorithms. Therefore, we opted for a Bottom-Up approach for our tests. This will be our steps:

1. Step 1 is to launch all the tests present on our application for see the current state of the application and check for any bugs or errors earlier.
2. Step 2 is to implement our new algorithm locally and write some unit tests (ie unit tests for classification algorithm). section 2.1.2). We use the written tests to validate that our new algorithm is working properly. Other manual tests on the new algorithm would be favorable for its validation at this stage.
3. Step 3 is to launch all the tests present in our application as well that regression tests to check and confirm that adding the new algorithm is correct without breaking our application and without generating errors.
4. Step 4 is to integrate our code into the application. As in our continuous integration plans. Integrating will automatically launch a series of tests regarding the code to be added. This step is the last step of validating the code and it will allow both checking the tests, but also to check the quality of the code that we want to integrate.

When adding the "KStart" classification algorithm, we’ll follow the 4 steps listed, we will integrate the new algorithm while ensuring the quality of the system already in place isn’t affected. It will be necessary to add the tests mentioned in step 2 (ie unit tests defined for the new algorithm and integration test) to the existing tests of our system. However, it won’t be necessary to update the quality plan, but we can always check that our objectives and criteria quality are still valid.

1. **Demonstrative Video**

Link to the video : <https://youtu.be/8-jKfq0db1I>

Link to Github : <https://github.com/fredsossa1/weka-3.8>

**References**

[1] EDUCBA, <https://www.educba.com/naive-bayes-algorithm/>

[2] GeekforGeeks, <https://www.geeksforgeeks.org/ml-linear-regression/>

[3] Wikipedia, <https://en.wikipedia.org/wiki/Farthest-first_traversal>

[4] Wikipedia, <https://en.wikipedia.org/wiki/Hierarchical_clustering>

[5] Course Slides from Moodle

[6] University of Waikato. (S.D) Weka 3: Data Mining Software in Java.

[En ligne]. Disponible : https://www.cs.waikato.ac.nz/ml/weka/