MIDDLE EAST

TECHNICAL UNIVERSITY

COMPUTER ENGINEERING

DEPARTMENT

Test Specification Report

● Myrzabek MURATALIEV	–	*1408665*
● Serdar DALGIÇ	–	*1448570*
● Haşim TİMURTAŞ	–	*1449149*
● Barış YÜKSEL	–	*1449826*

Table Of Contents

1. INTRODUCTION

Wekarel is the parallel implementation of popular machine learning software Weka, and focuses on achieving data and task parallelism while evaluating clusters. Wekarel, compatible with 3.4 version of Weka, is first attempt when compared with any conjugate software based on Weka. Details of the project can be found in final design report[1].

1.1. Goals and Objectives

Our goals for testing the process of the project “Wekarel” are: achieving a bug-free(as much as we can achieve), high performance and a consistent-to-standards product. As a result of this, testing is one of the most important parts of our process. Testing is always a challenging process, however it helps to deliver usable systems.

Starting from the project definition, in this report, we will describe the procedures to test the components making up the system and the overall system as a whole. The tests will verify that not only the system operates correctly, but also it runs fast and is platform independent.

1.2. Scope of Document

This document briefly describes the testing process of the project “Wekarel”. As in fact the testing process is continuous from the starting date of the project to the end date, this document also includes some information about how the testing has been done up till now in the project. However the main aim and scope of this document is to present the testing plan for the remaining part of the project in which the implementation is considered to be nearly complete.

1.3. *Statement of Testing Plan Scope*

Testing process of the project “Wekarel” includes unit testing, integration testing, validation testing, performance testing, stress testing, alpha and beta testing.

Unit Testing: This testing strategy involves testing process of modules as if they are standalone parts.

Data Testing: With that kind of testing, we guarantee the types of data our application accepts.

Integration Testing: In that phase of testing, we mainly focused on the integrability of the modules we have developed, their main relationships between each of these modules and untouched parts of the Weka code.

Validation Testing: This test is applied to see if the requirement and design goals are achieved.

Performance Testing: Includes testing of methods used to improve the speed and the quality of the project.

Stress Testing: The test to see the maximum capability of the *Wekarel* and its modules by means of data length,

Alpha Testing: Testing of *Wekarel* by someone other than the group members, working on the same project.

Beta Testing: The test applied by end-users to reach a more stable version.

1.4. *Major Constraints*

We had several different constraints in our testing plan. For our project most important constraint was time since we had to finish the final package on time while we were testing our software. Another constraint of our testing plan was the number of staff. We had 4 group members and it made difficult to do all tests in time. Since our project could not work on a standard computer but a high performance computer – NAR; we had to bow to the hardware constraints too.

1.4.1. Time

Since *Wekarel* should have been finished due to mid of June, time was the main constraint for us. We had a schedule for the testing phase, therefore the deadlines of each specific work were known beforehand. If we managed to obey the deadlines according to schedule, our project was released after proper testing and in time.

1.4.2. Staff

All members of the testing process were just the 4 of the senior design project group, which are also responsible for developing the main parts of the project. We tried to overcome this problem with assigning Test-Responsibles to the parts that they have not dealed with. However the lack of extra stuff in testing became a major constraint for the testing process.

1.4.3. Hardware

The fundamental work here is being used on multicore machines. Due to several reasons also related with the used software, the workplace of the testing process has been limited to restricted to ssh-tunneling to NAR from department's network.

1.4.4. Software

We are using PJ library for parallelization parts of the project. However, due to constraints related to PJ, we were not able to run our test schemas on distributed computing over mutiple computers/cores except NAR. This constraint also affected us in terms of Hardware constraints.

2. TESTING STRATEGY AND PROCEDURES

2.1. *Unit Testing*

While developing our project by proceeding module by module, we are making sure that every module is processing correctly so that when these modules are combined, we can be sure that there are no problems in the basement. We believe that the modules can work in harmony more easily by proceeding in this bottom up manner. All the modules are tested in a black box manner, by observing the input that is provided to the module and the output or the effect produced as a result. If a problem occurs during black box testing and it can be handled, we are also planning to make use of white box testing.

2.2. *Data Testing*

The generic file type which Weka uses is ARFF (Attribute-Relation File Format) , and has these specifications:

i. *The name of the relation should be specified at the first line, following the tag "@relation".*
ii. *The name of each attribute should be indicated at the following lines, following the tag "@attribute". The data type of the attribute (real, nominal, etc.) should also be denoted at that line.*
iii. *Each data (Instance object) should be written at the lines followed by the tag "@data".*

```
@RELATION iris

  @ATTRIBUTE sepallength NUMERIC

  @ATTRIBUTE sepalwidth  NUMERIC

  @ATTRIBUTE petallength NUMERIC

  @ATTRIBUTE petalwidth  NUMERIC

  @ATTRIBUTE class     {Iris-setosa,Iris-versicolor,Iris-virginica}
@DATA

  5.1,3.5,1.4,0.2,Iris-setosa

  4.9,3.0,1.4,0.2,Iris-setosa

  4.7,3.2,1.3,0.2,Iris-setosa

  4.6,3.1,1.5,0.2,Iris-setosa

  5.0,3.6,1.4,0.2,Iris-setosa

  5.4,3.9,1.7,0.4,Iris-setosa

  4.6,3.4,1.4,0.3,Iris-setosa
```

Figure 1 : Sample ARFF file contents

Most of the methods we have parallelized use the ARFF file as the dataset, and we drew up the built-in datasets and the one we have generated so that they all obey the specifications of an ARFF file. Additionally, some classes in which low – level parallelization is provided (such as *Utils* class under *weka.core* package) do not use the specific *ARFF* file, since they do not deal with the *Instances* objects. In these cases, only the “data” part of the *ARFF* file had been taken into consideration, and we assured that the data would be given in that form.

2.3. *Integration Testing*

During the development, one of the main integration testing was conducted when PJ library was integrated to the project. When integration was completed, PJ's interaction with rest of the project was tested. Every time, a module with PJ related methods is integrated to the project, simple integration testing is conducted. Additionally, PJ library has many easy to use testing methods which were implemented in constructing PJ library itself and some of them are really helping us also in integration testing part. Besides, all defined classess of Weka code aquires main function which enables us to test all methods implemented in that class. Many small but cardinal mistakes were discovered and eliminated during integration testings. For example, in one of the testings it was detected that K-Means algorithm was not running properly as one of the parallelized methods of Utils package also defined Comm variable. Shortly, integration testing is an important test that all new modules have to pass.

2.4. *Validation Testing*

Validation Testing is the different set of activities which ensures that the software that we are developing is traceable to the requirements that we have already stated in our requirement analysis report. Validation Testing is the responsibility of the tester, and all the tests done in block box testing will be treated as validation tests. Since we do not have a specific test group we tried to simulate this situation in low level. Parts being developed by one of us also tested by the other ones. We have stated requirements validation below:

2.4.1. User Interface

We have planned to have a graphical user interface for project that we are developing. However later since we have seen that is not in our main focus. We left it to be (may be) developed after our project duration as an open source. So we could not do any test on that issue.

2.4.2. Input Output Requirements

We have already know about input output specifications of weka. So while developing wekarell we also tried to obey weka's input output specifications. We have done our test for ARFF file type.

2.4.3. Parallel Approach

During all parts of our development process, we needed to keep in mind how a parallel program works. That notion has also helped us choosing and eliminating development tools for our project. The main argument of parallel development is speed-up. Our test shows us that although for test files with small amount of data we have a low speed-up; we could increase speed-up by using larger dataset files. Since that speed-up is one of our main concerns, it was effective for us to choose which modules should be parallellized.

2.5. *High – Order Testing*

Moreover we are planning the following high level tests for the project *Wekarel*:

2.5.1. Performance Tests

In order to obtain a reliable value of the speed – up and the running time of the algorithms we have parallelized, we have executed these parallel algorithms numerous times and took the mean value of the running time of these executions into consideration. Also, since *Weka* is a data mining tool, we should deal with large amounts of data, therefore conduct the tests usually with data larger than 1M. Working with large amount of data also provided us a higher speedup in the parallelization.

2.5.2. Stress Tests

We have tested parallelized algorithms with extreme amounts of data, and there were cases that the Java Virtual Machine crashed due to the insufficiency of its heap size. Even if we modified the arguments of the virtual machine to increase its maximum permitted heap size, we could not avoid the crash. This problem is purely Java and JVM related; it has nothing to do with our parallelized codes.

2.5.3. Alpha and Beta Tests

All the tests done during the development of *Wekarel* are counted to be in the scope of Alpha test. As inherent to Beta tests, main testers are expected to be customers. But, in our project it is difficult to say that it has a specific customer. All testers are expected to have enough understanding of Weka tool and parallel programming concepts. Additionally, uniqueness of requirements is making it impossible to conduct testing on this project modules for customers. In other words, in order to conduct a testing, person must have an access to multi processing systems like NAR and this system must have all important libraries like PJ. As a result of these obstacles in Beta testings we were unable to conduct it in our testing phase. Alpha tests are the only tests conducted throughout development of the project.

3. TEST RECORD KEEPING AND LOG

As our testing process is taking along with the development process and the testing staff is restricted to the team members, these facts have also affected the record keeping and logging processes.

3.1. *Test Report Forms*

We didn't design any kind of special forms for testing and getting feedback for testers (alpha and beta). We use *scicomp google group* for general communication between the team members, and this group also reveals as a bug tracking tool and feedback receiver for the testing members. The reason why the team has decided to do so is the general behavior of the inherited project Weka, which has no bug-tracking system but a users mailing – list. Responsible group members have already been tracking the users mailing list and after the release of *Wekarel,* as the project will be related to Weka project and users would be conducting via Weka users mailing – list, possible feedbacks and testing records are expected to be reached from this list.

Another reason why no types of testing forms are created is the limitation of the testing process only to the development members. All developers were already reporting the results of their white box tests and no need for individual testing forms has appeared during the testing process.

4. TESTING TOOLS AND ENVIRONMENTS

- **Eclipse jee-ganymede-SR2-linux-gtk-x86_64 :** For Java Code Developing and debugging.
- **Weka 3-4-14 Tests :** Using Black box and beta tests that are already written for Weka Tool.
- **PJ Tests :** Using Black box and beta tests that are already written for Parallel Java(PJ) Library.

5. TESTING RESOURCES AND STAFFING

The only resource required for testing is a PC that has the capability of SSH tunneling and X forwarding to HPC Machine NAR, in which *eclipse* can be run and allow the possible conditions for the environment to test.

As the development process weren't seperated in sharp lines between the team members, the testing process has the properties too. The point that has been noted is assigning the testing processes of the developed modules to the members who are not involved in the coding of these development parts. So that more individuals have the chance to take an eye on the code and this situation allows the testing process to be staffed in a strong way.

6. REFERENCES

[1] http://senior.ceng.metu.edu.tr/2009/scicomp/docs/SciComp_FinalDesign_v0.2.pdf