Library for Associative Classification (LAC)

User Manual
LAC version 0.2.0
November 20, 2019

Francisco Padillo
Jose Maria Luna
Sebastian Ventura

Contents

1 Introduction **1**

2 Getting and installing LAC software **3**

2.1 Downloading LAC 3
2.2 Building LAC 9
2.3 Installing LAC 15

3 Algorithms included in LAC **16**

3.1 CBA 16
3.2 CBA2 17
3.3 CMAR 18
3.4 CPAR 18
3.5 MAC 20
3.6 ACCF 20
3.7 L3 21
3.8 ACN 22
3.9 ADT 23
3.10 ACAC 24

4 LAC Software for users **25**

4.1 Input of datasets 25
4.2 Reports of algorithms 27
4.3 Configuration 29
4.4 Running an existing algorithm 31
4.5 Using the framework to automate the experimental study 32

5 LAC Software for developers **37**

5.1 Structure of LAC packages/classes 37

5.2 Adding a new input format 39

5.3 Developing a new algorithm 42

5.4 Adding a new type of report 47

6 Reporting bugs **50**

A YAML format **52**

Bibliography **54**

Chapter 1

Introduction

As technology advances, very large quantities of data are constantly being generated. The production and collection of those large quantities of data have encouraged the analysis and the extraction of useful but hidden information. Data analysis is the term used to refer to those techniques [11]. In data analysis, two different tasks are considered: descriptive tasks, which depict intrinsic and important properties of data [3]; and predictive tasks, which predict output variables for unseen data [11] by learning a mapping between a set of input variables and the output variable. Focusing on the last one, different methodologies can be considered: rule-based systems [11], decision trees [23] and support vector machines [7], just to list a few. From all these methodologies, rule-based classifiers provide a high-level of interpretability and, therefore, classification results can be explained since rules tend to be easily understood and interpreted by the end-user. Additionally, different research studies [2] have demonstrated that associative classifier are competitive with regard to other methodologies [4].

Classification based on association rule mining, generally known as Associative Classification (AC), integrates a descriptive task (association rule mining [3]) in the process of inferring a new classifier [17]. Recent studies [2] have shown that AC has the following advantages over traditional classification approaches: 1) Accuracy [17], models in AC are often capable of building efficient and accurate classification systems since in the training phase they leverage association rule discovery methods that find all possible relationships among the attribute values; 2) Usability [21], unlike decision tree approaches, AC does not require to redraw the whole model when the rule set is updated and tuned; 3) Readability [2], the final model of AC comprises a simple set of rules that allow the end-user to easily understand and interpret the results. Nonetheless, it is important to remark that AC models are not often as accurate as black-box models [11].

However, in spite of AC is really interesting for the research community, to the best of our knowledge there is not any available library or tool [10, 26] covering the full taxonomy of AC [2]. Existing tools have its own restrictions with regard to input format, quality metrics, and the inability to automate and parallelize experimental studies. Additionally, nowadays with the rise of the replication crisis new tools are required to improve reproducibility among research studies [19]. Thus, the main contributions of Library for Associative Classification (LAC) could be summarized in four points: 1) It covers the whole AC taxonomy; 2) It includes plenty of quality measures to quantify not only the quality but also the intrepretability; 3) It is easy to be used with multiple input data formats (arff [10], keel dat [26] and csv [25]); 4) It enables to fully automate and parallelize the experimental studies.

The rest of this manual is organized as follows. Section 2 describes how to get and install LAC software in your own computer. Section 3 describes all the algorithms included in LAC and its configurations. Section 4 describes how to correctly use LAC. Section 5 is mainly directed to developers interested in extending LAC. Finally, Section 6 describe what is the best way of reporting bugs.

Chapter 2

Getting and installing LAC software

LAC is publicly available at its git repository hosted on Github. The best place to find the very last version of LAC is its repository, from there you could find a tag for each released version. Each version is accompanied with its respectively user manual, hosted on the same repository as the software.

Aiming at solving the well-known problem of dealing with dependencies, also known as "dependency hell", LAC follows Semantic Versioning 2.0.0 (more information could be found at `https://semver.org/`). This standard could be summarized as given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when incompatible API changes are made.
- MINOR version when new functionality in a backwards-compatible manner are added.
- PATCH version when backwards-compatible bug fixes are done.

Following this convention, the very first version of LAC is released under the 0.2.0 version and it is publicly available at `https://github.com/kdis-lab/lac/releases/tag/v0.2.0`.

2.1 Downloading LAC

LAC could be downloaded using two different ways: using git as command line or directly from the webpage. Both options are described below.

Using git

Next, the installation of git is described for each supported operative system and a complete explanation on how to download LAC is also carried out.

- GNU-Linux. In this operating system git packages could be installed using the package manager of your distribution. For Debian based systems, it could be installed using:

  ```
  # apt update && apt install git-core
  ```

  Similarly, for Fedora based systems, it could be installed running:

  ```
  # yum install git-core
  ```

- MAC OS X. If XCode was previously installed, git may already be installed. Anyway, git could be installed using Homebrew running the following command.

  ```
  # brew install git
  ```

- Windows. For this operating system there is a software called *Git for windows* which enables to easily install Git. It can be publicly found on its website (`https://gitforwindows.org/`). From this website (see Figure 2.1), the *Download* button is clicked.

  This button automatically detects which operating system are being used and download the best possible version. In cases where the detector cannot automatically

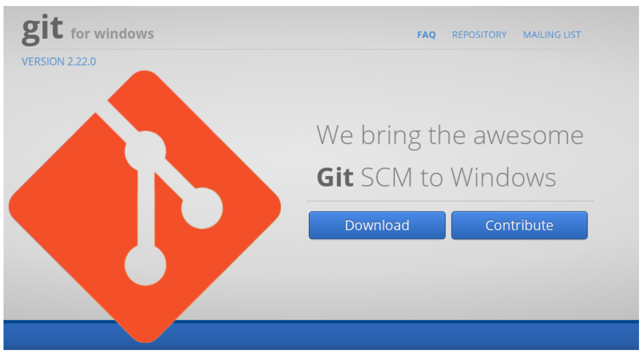
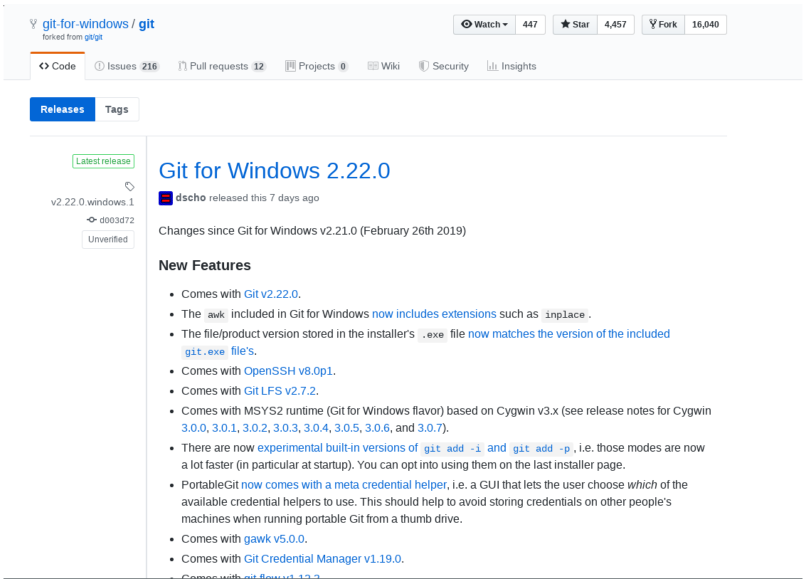


Figure 2.1: Git for Windows website. The download button has to be clicked to obtain the latest version.

check which is the best version, it redirects to the Github releases page for this software, where all the available versions could be found. At the time of writing this document, the latest available version is 2.22.0 (See Figure 2.2).

The binary installer could be found under the Assets drop-down (See Figure 2.3). *exe* files are recommended since they are more easy to install.



Git for Windows 2.22.0

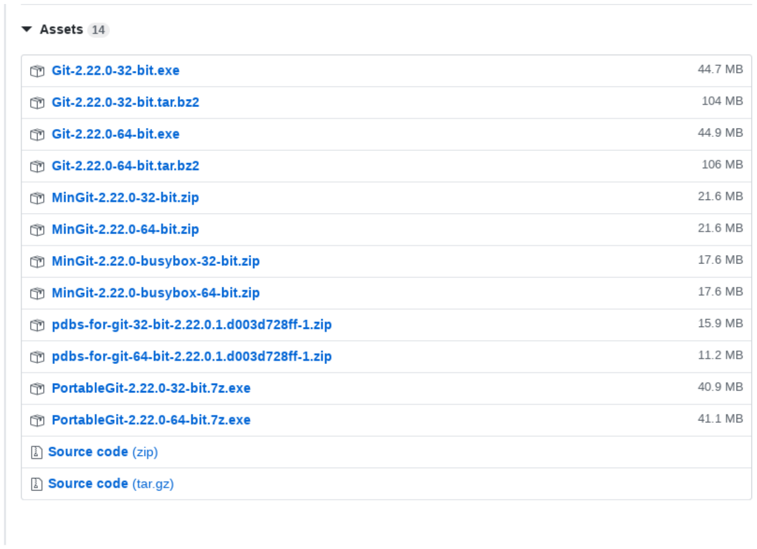
dscho released this 7 days ago

Changes since Git for Windows v2.21.0 (February 26th 2019)

New Features

- Comes with Git v2.22.0.
- The `awk` included in Git for Windows now includes extensions such as `inplace`.
- The file/product version stored in the installer's `.exe` file now matches the version of the included `git.exe` file's.
- Comes with OpenSSH v8.0p1.
- Comes with Git LFS v2.7.2.
- Comes with MSYS2 runtime (Git for Windows flavor) based on Cygwin v3.x (see release notes for Cygwin 3.0.0, 3.0.1, 3.0.2, 3.0.3, 3.0.4, 3.0.5, 3.0.6, and 3.0.7).
- There are now experimental built-in versions of `git add -i` and `git add -p`, i.e. those modes are now a lot faster (in particular at startup). You can opt into using them on the last installer page.
- PortableGit now comes with a meta credential helper, i.e. a GUI that lets the user choose *which* of the available credential helpers to use. This should help to avoid storing credentials on other people's machines when running portable Git from a thumb drive.
- Comes with gawk v5.0.0.
- Comes with Git Credential Manager v1.19.0.

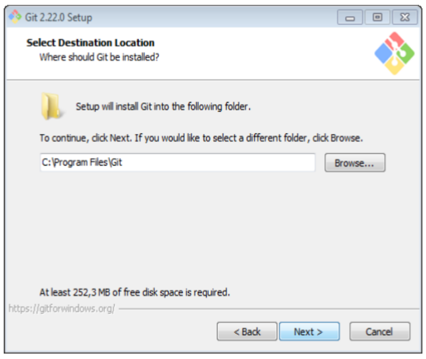
Figure 2.2: Git for Windows Github website. All the publicly available versions are listed on this page.



Git-2.22.0-32-bit.exe	44.7 MB
Git-2.22.0-32-bit.tar.bz2	104 MB
Git-2.22.0-64-bit.exe	44.9 MB
Git-2.22.0-64-bit.tar.bz2	106 MB
MinGit-2.22.0-32-bit.zip	21.6 MB
MinGit-2.22.0-64-bit.zip	21.6 MB
MinGit-2.22.0-busybox-32-bit.zip	17.6 MB
MinGit-2.22.0-busybox-64-bit.zip	17.6 MB
pdbs-for-git-32-bit-2.22.0.1.d003d728ff-1.zip	15.9 MB
pdbs-for-git-64-bit-2.22.0.1.d003d728ff-1.zip	11.2 MB
PortableGit-2.22.0-32-bit.7z.exe	40.9 MB
PortableGit-2.22.0-64-bit.7z.exe	41.1 MB
Source code (zip)	
Source code (tar.gz)	

Figure 2.3: Git for Windows Github website. 64 or 32 bit may be downloaded in function of your system.

Once installer has been downloaded, it has to be started making double click. The Git Setup wizard screen starts. Licence should be read before continuing, and the Next button has to be clicked. Several steps are shown to configure all the options of Git for Windows (See Figure 2.5). Default options are enough, and no change is required.



Select Destination Location
Where should Git be installed?

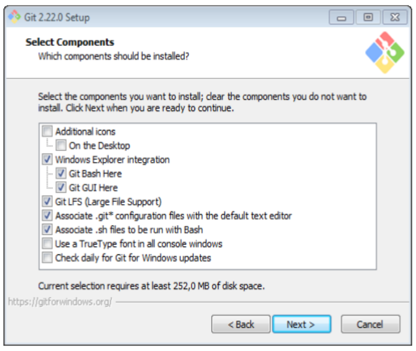
Setup will install Git into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Git

At least 252,3 MB of free disk space is required.

(a) Step 1. Path of installation.



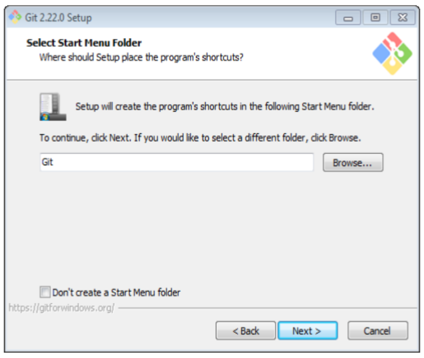
Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Additional icons
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 252,0 MB of disk space.

(b) Step 2. Components of installation.



Select Start Menu Folder
Where should Setup place the program's shortcuts?

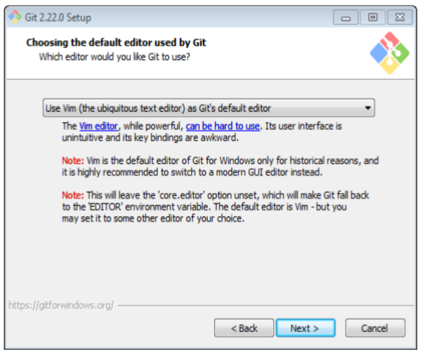
Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

Git

- [ ] Don't create a Start Menu folder

(c) Step 3. Menu folder.



Choosing the default editor used by Git
Which editor would you like Git to use?

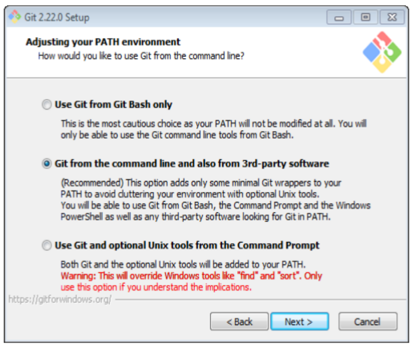
Use Vim (the ubiquitous text editor) as Git's default editor

The Vim editor, while powerful, can be hard to use. Its user interface is unintuitive and its key bindings are awkward.

Note: Vim is the default editor of Git for Windows only for historical reasons, and it is highly recommended to switch to a modern GUI editor instead.

Note: This will leave the 'core.editor' option unset, which will make Git fall back to the 'EDITOR' environment variable. The default editor is Vim - but you may set it to some other editor of your choice.

(d) Step 4. Default editor.



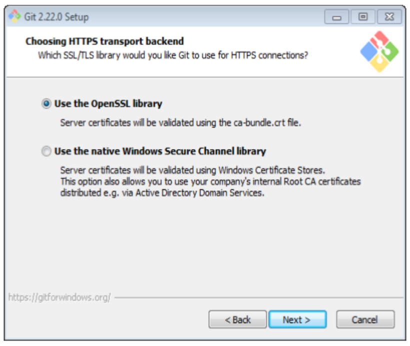
Adjusting your PATH environment
How would you like to use Git from the command line?

○ **Use Git from Git Bash only**
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

● **Git from the command line and also from 3rd-party software**
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

○ **Use Git and optional Unix tools from the Command Prompt**
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

(e) Step 5. PATH Environment.

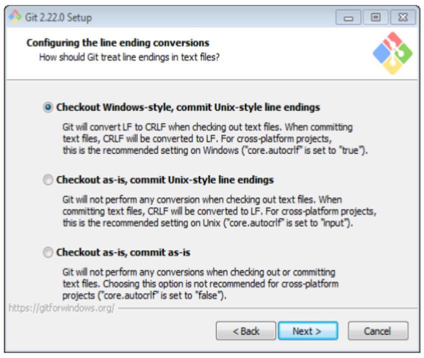


Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?

● **Use the OpenSSL library**
Server certificates will be validated using the ca-bundle.crt file.

○ **Use the native Windows Secure Channel library**
Server certificates will be validated using Windows Certificate Stores. This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

(f) Step 6. HTTPS configuration.



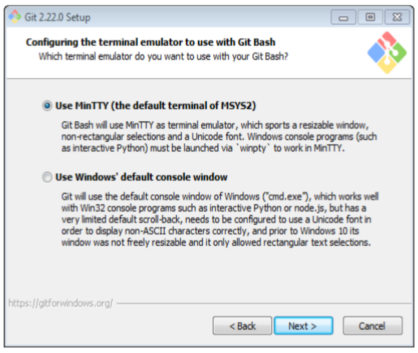
Configuring the line ending conversions
How should Git treat line endings in text files?

● **Checkout Windows-style, commit Unix-style line endings**
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

○ **Checkout as-is, commit Unix-style line endings**
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

○ **Checkout as-is, commit as-is**
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

(g) Step 7. Ending conventions.



Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?

● **Use MinTTY (the default terminal of MSYS2)**
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

○ **Use Windows' default console window**
Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

(h) Step 8. Terminal emulator.

Figure 2.5: Different steps of the installation of Git for Windows.

Once git is installed, to check if it was correctly installed the following command may be executed. In windows it should be run on the Command prompt of windows (cmd). In GNU-Linux or MAC OS X it could be run on whatever terminal emulator is being used.

```
$ git --version
```

Finally, LAC could be easily downloaded running:

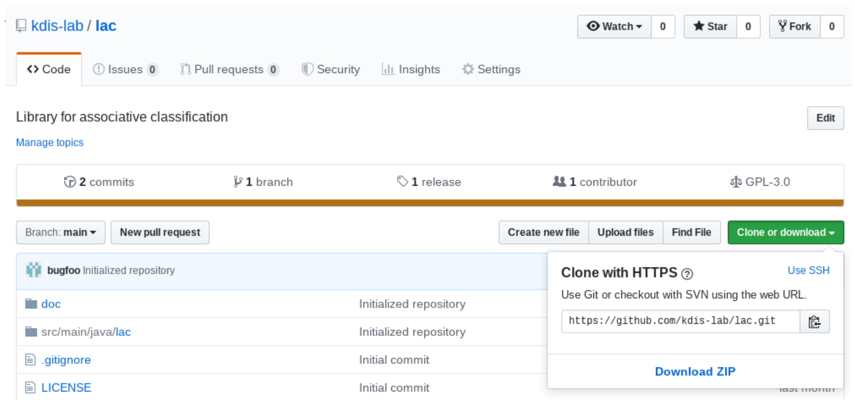
```
$ git clone --branch v0.2.0 https://github.com/kdis-lab/lac.git
```

It will create a directory on the current path with the name of *lac*. This directory contains the source code and the user manual for the version previously specified. Thus, the current path has to be changed to the new created directory using the command *cd*.

```
$ cd lac
```

Using Github web

Github provides a direct link to download LAC without requiring to use git. It is much more easy to download, but it does not enable to fetch latest updates easily but this whole process has to be repeated for each new released version. LAC could be downloaded visiting the Github repository publicly available at `https://github.com/kdis-lab/lac`, and clicking on the green button with the text *Clone or download*. Then, a small pop-up opens, showing a *Download ZIP* button (See Figure 2.6). After clicking on that button, a compressed file is downloaded with all the content of the repository. ZIP file only has to be extracted to obtain all the content of the repository.



Library for associative classification

bugfoo Initialized repository

doc	Initialized repository
src/main/java/lac	Initialized repository
.gitignore	Initial commit
LICENSE	Initial commit

Use Git or checkout with SVN using the web URL.

```
https://github.com/kdis-lab/lac.git
```

Figure 2.6: Using Github website to download LAC, clicking on the *Clone or download* button.

2.2 Building LAC

As it was previously stated LAC has been developed using Java (1.8+) and it makes use of Maven to install all the dependencies. Thus, the first step to build LAC is to install Java JDK (1.8+).

- GNU-Linux. In this operating system Java could be installed using openjdk package. For Debian based systems, it could be installed using its package manager as follows.

  ```
  # apt update && apt install openjdk-8-jdk
  ```

  Likewise, for Fedora based systems, it could be installed running the following command.

  ```
  # yum install java-1.8.0-openjdk
  ```

- MAC OS X. Java could be easily installed using Homebrew. As Java is not open source, *cask* has to be installed to be used together with Homebrew.

  ```
  # brew tap caskroom/cask
  ```

  Once *cask* has been successfully installed, Java could be installed using *brewcask* running the following command.

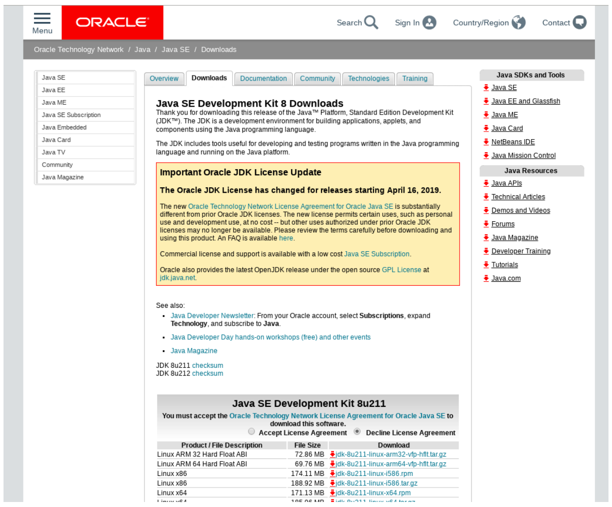
  ```
  # brew cask install caskroom/versions/java8
  ```

- Windows. Installer for Java are available at the oracle website (`https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`). In this website all the versions could be found (See Figure 2.7).

  At the time of writing this document, the latest version of Java 1.8+ is Java SE Development Kit 8u211. License should be accepted before downloading. 32 or 64 bit may be downloaded in function of your system (See Figure 2.8).

  After downloading, installer has to be run making double-click. It opens a new installation setup wizard as shown in Figure 2.9, where *Next* button has to be clicked.

  Installer also asks for the path of installation, as shown in Figure 2.10, there are no need to change that, thus default path is used and click on *Next* to continue the installation.



Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Important Oracle JDK License Update

The Oracle JDK License has changed for releases starting April 16, 2019.

The new Oracle Technology Network License Agreement for Oracle Java SE is substantially different from prior Oracle JDK licenses. The new license permits certain uses, such as personal use and development use, at no cost -- but other uses authorized under prior Oracle JDK licenses may no longer be available. Please review the terms carefully before downloading and using this product. An FAQ is available here.

Commercial license and support is available with a low cost Java SE Subscription.

Oracle also provides the latest OpenJDK release under the open source GPL License at jdk.java.net.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u211 checksum
JDK 8u212 checksum

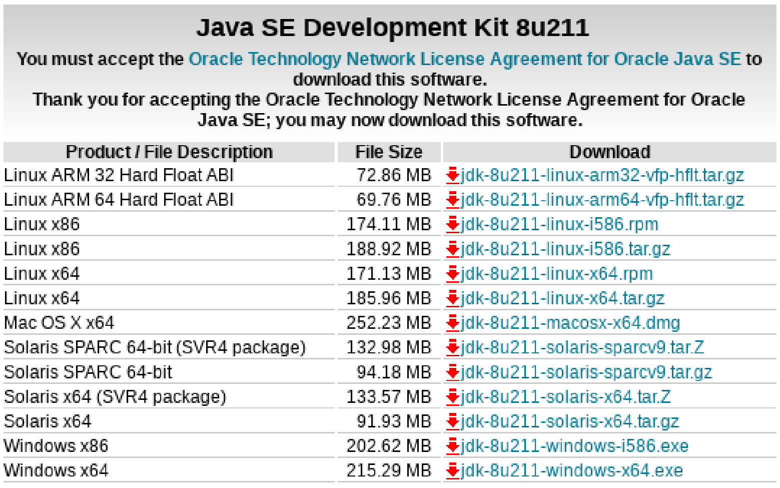
Java SE Development Kit 8u211

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.86 MB	jdk-8u211-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.76 MB	jdk-8u211-linux-arm64-vfp-hflt.tar.gz
Linux x86	174.11 MB	jdk-8u211-linux-i586.rpm
Linux x86	188.92 MB	jdk-8u211-linux-i586.tar.gz
Linux x64	171.13 MB	jdk-8u211-linux-x64.rpm

Figure 2.7: Oracle Website. All the versions are listed on this website.

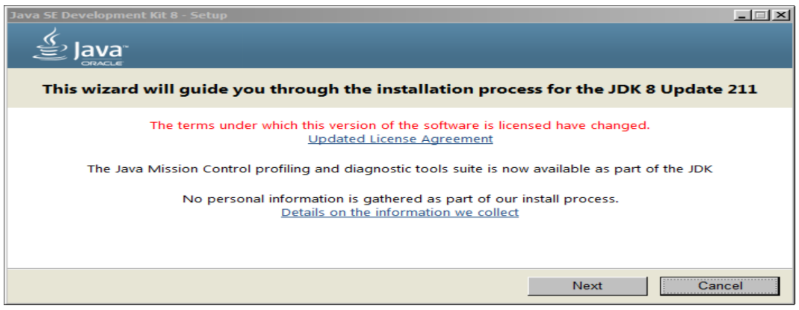


Java SE Development Kit 8u211

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.
Thank you for accepting the Oracle Technology Network License Agreement for Oracle Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.86 MB	jdk-8u211-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.76 MB	jdk-8u211-linux-arm64-vfp-hflt.tar.gz
Linux x86	174.11 MB	jdk-8u211-linux-i586.rpm
Linux x86	188.92 MB	jdk-8u211-linux-i586.tar.gz
Linux x64	171.13 MB	jdk-8u211-linux-x64.rpm
Linux x64	185.96 MB	jdk-8u211-linux-x64.tar.gz
Mac OS X x64	252.23 MB	jdk-8u211-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	132.98 MB	jdk-8u211-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.18 MB	jdk-8u211-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.57 MB	jdk-8u211-solaris-x64.tar.Z
Solaris x64	91.93 MB	jdk-8u211-solaris-x64.tar.gz
Windows x86	202.62 MB	jdk-8u211-windows-i586.exe
Windows x64	215.29 MB	jdk-8u211-windows-x64.exe

Figure 2.8: Java SE Development Kit 8u211 is the latest version at the time of writing this document.



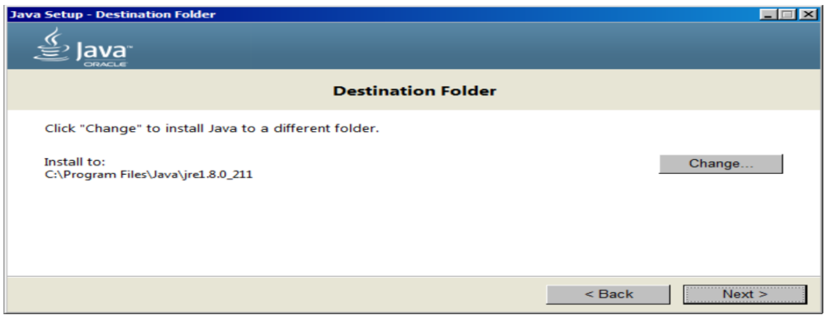
This wizard will guide you through the installation process for the JDK 8 Update 211

The terms under which this version of the software is licensed have changed.
Updated License Agreement

The Java Mission Control profiling and diagnostic tools suite is now available as part of the JDK

No personal information is gathered as part of our install process.
Details on the information we collect

Figure 2.9: Installation of Java.


Java Setup - Destination Folder

Destination Folder

Click "Change" to install Java to a different folder.

Install to:
C:\Program Files\Java\jre1.8.0_211

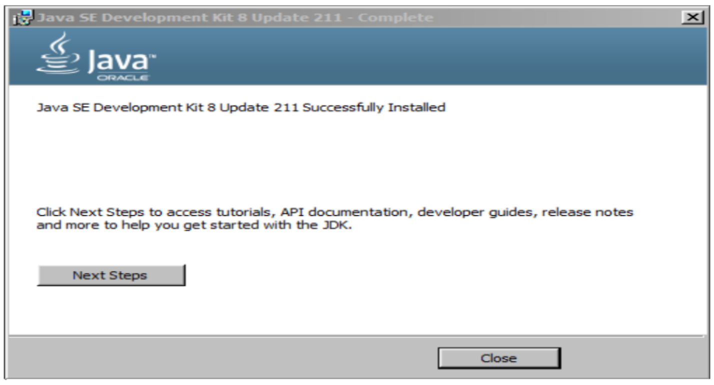
Figure 2.10: Installation of Java, path of installation is asked. Default path may be used.

Once installer has finished its work, it shows a message of success as that shown in Figure 2.11.

After that, an environment variable has to be created. Environment variable could be created making click on *Start* button to start *Control Panel* (See Figure 2.12). Then, *System and Security* should be clicked to then click on *System*. Once there, *Advanced system settings* should be clicked (see Figure 2.13).

It opens a new configuration windows (as is shown in Figure 2.14). In that window, *Advanced Tab* has to be clicked, and then on the *Environment Variables button*.

It opens a new configuration windows, where the $JAVA\_HOME$ environment variable has to be created with the path of the installation (See Figure 2.15).


Java SE Development Kit 8 Update 211 - Complete

Java SE Development Kit 8 Update 211 Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Figure 2.11: Installation of Java, installer has finished its work and Java is available.

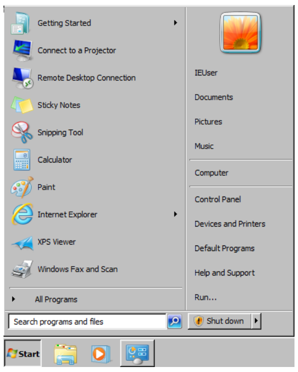
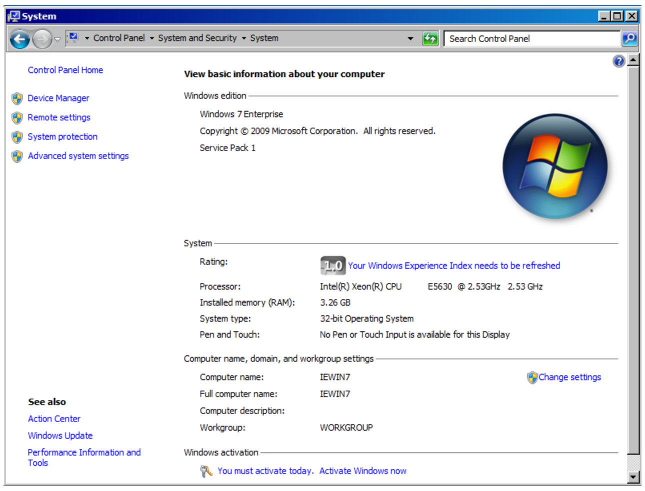


Figure 2.12: Click on Start to open Control Panel.



View basic information about your computer

Windows edition

Windows 7 Enterprise

Copyright © 2009 Microsoft Corporation. All rights reserved.

Service Pack 1

System

Rating: 1,0 Your Windows Experience Index needs to be refreshed

Processor: Intel(R) Xeon(R) CPU E5630 @ 2.53GHz 2.53 GHz

Installed memory (RAM): 3.26 GB

System type: 32-bit Operating System

Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name: IEWIN7 Change settings

Full computer name: IEWIN7

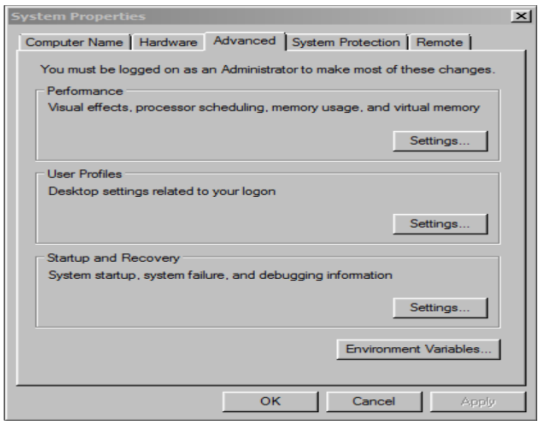
Computer description:

Workgroup: WORKGROUP

Windows activation

You must activate today. Activate Windows now

Figure 2.13: Control Panel in System settings, Advanced system settings from the left menu should be clicked.



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

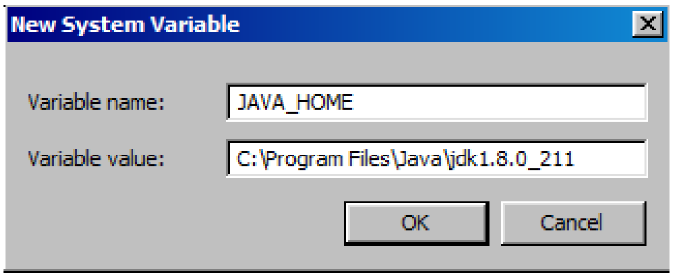
User Profiles

Desktop settings related to your logon

Startup and Recovery

System startup, system failure, and debugging information

Figure 2.14: System properties are opened, and the Environment Variables button should be clicked.


New System Variable

Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.8.0_211

Figure 2.15: Installation of Java. Adding a new system environment variable for Java.

Installation could be checked running the following command, it should be noted that at least version 1.8 should be returned.

```
$ java -version
openjdk version "1.8.0_212"
OpenJDK Runtime Environment (build 1.8.0_212-8u212-b03-2~deb9u1-b03)
OpenJDK 64-Bit Server VM (build 25.212-b03, mixed mode)
```

Once Java has been successfully installed, Maven need to be installed to manage all the dependencies of LAC.

- GNU-Linux. It could be installed using the package manager of your distribution. For debian based system it could be installed running.

  ```
  # apt update && apt install maven
  ```

  Likewise, it could be installed on Fedora based systems using yum.

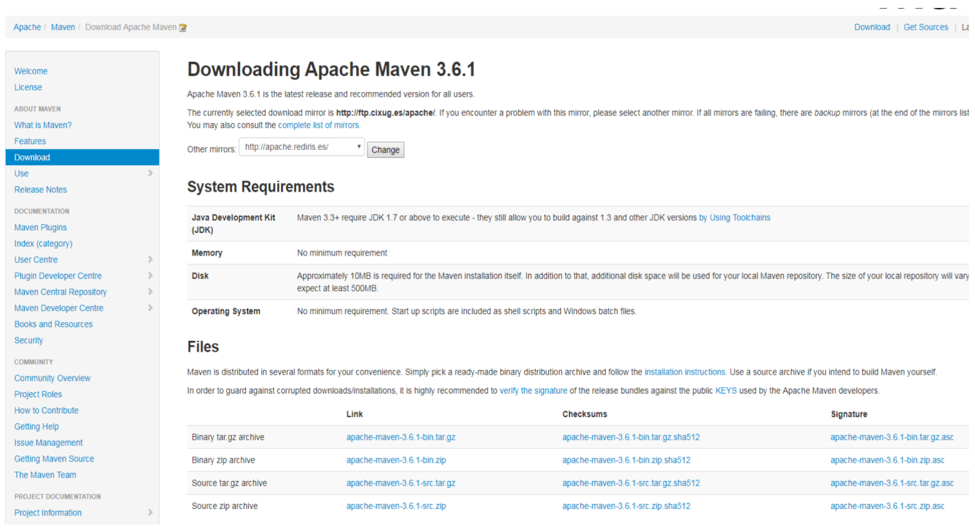
  ```
  # yum install maven
  ```

- MAC OS X. Maven could be easily installed using Homebrew running the following command.

  ```
  # brew install maven
  ```

- Windows. It could be downloaded from the official website of Maven at `https://maven.apache.org/download.cgi`. Binary zip archive may be downloaded (see Figure 2.16)

  Once Maven is downloaded, ZIP archive has to be extracted on the directory of installation. For instance, in this tutorial it is installed on $C:\backslash apache\text{-}maven$, drive



Apache / Maven / Download Apache Maven

Download | Get Sources | La

Welcome
License
ABOUT MAVEN
What is Maven?
Features
Download
Use
Release Notes
DOCUMENTATION
Maven Plugins
Index (category)
User Centre
Plugin Developer Centre
Maven Central Repository
Maven Developer Centre
Books and Resources
Security
COMMUNITY
Community Overview
Project Roles
How to Contribute
Getting Help
Issue Management
Getting Maven Source
The Maven Team
PROJECT DOCUMENTATION
Project Information

Downloading Apache Maven 3.6.1

Apache Maven 3.6.1 is the latest release and recommended version for all users.

The currently selected download mirror is **http://ftp.cixug.es/apache/**. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are *backup* mirrors (at the end of the mirrors list
You may also consult the complete list of mirrors.

Other mirrors: http://apache.rediris.es/ Change

System Requirements

Java Development Kit (JDK)	Maven 3.3+ require JDK 1.7 or above to execute - they still allow you to build against 1.3 and other JDK versions by Using Toolchains
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, additional disk space will be used for your local Maven repository. The size of your local repository will vary expect at least 500MB.
Operating System	No minimum requirement. Start up scripts are included as shell scripts and Windows batch files.

Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

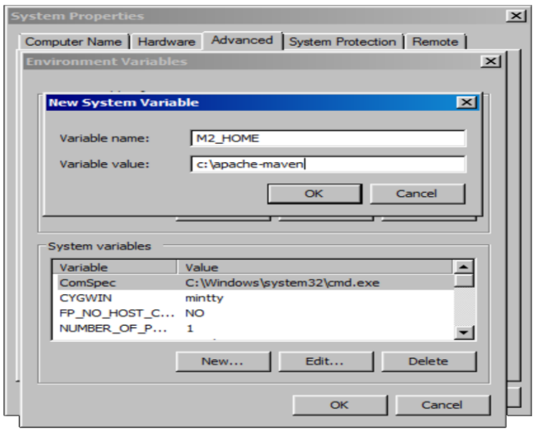
In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.6.1-bin.tar.gz	apache-maven-3.6.1-bin.tar.gz.sha512	apache-maven-3.6.1-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.1-bin.zip	apache-maven-3.6.1-bin.zip.sha512	apache-maven-3.6.1-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.1-src.tar.gz	apache-maven-3.6.1-src.tar.gz.sha512	apache-maven-3.6.1-src.tar.gz.asc
Source zip archive	apache-maven-3.6.1-src.zip	apache-maven-3.6.1-src.zip.sha512	apache-maven-3.6.1-src.zip.asc

Figure 2.16: Maven website. It could be downloaded clicking on apache-maven-3.6.1-bin.zip.

letter may change in your system. After extracting, an environment variable has to be created. Environment variable could be created as it was previously described for $JAVA\_HOME$. For this particular case, the $M2\_HOME$ environment variable has to be created with the path of the installation (See Figure 2.17).

To end this step, $PATH$ variable has to be edited to add the directory of binaries of maven, that is, $;C:\backslash apache\text{-}maven\backslash bin$. It should be noted that a ; character has been added at the beginning to separate among values in this $PATH$ variable (See Figure 2.18). These captures have been obtained on Windows 7, in other versions of windows may briefly change.

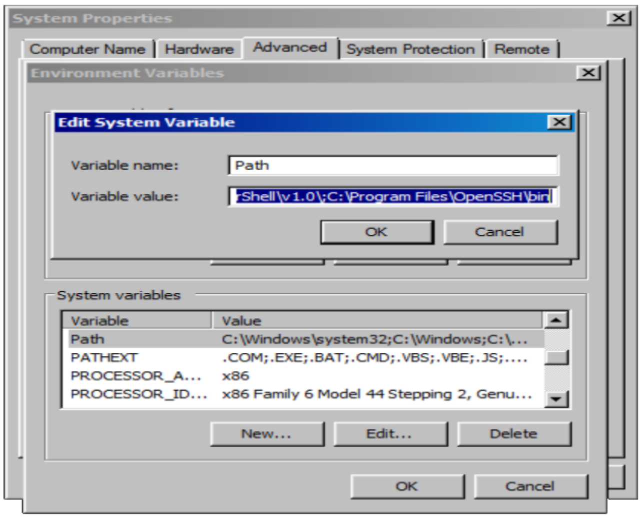


Variable name: M2_HOME
Variable value: c:\apache-maven

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
CYGWIN	mintty
FP_NO_HOST_C...	NO
NUMBER_OF_P...	1

Figure 2.17: Adding a new environment variable for Maven.



Variable name: Path

Variable value: Shell\v1.0\;C:\Program Files\OpenSSH\bin

System variables

Variable	Value
Path	C:\Windows\system32;C:\Windows;C:\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	x86
PROCESSOR_ID...	x86 Family 6 Model 44 Stepping 2, Genu...

Figure 2.18: Installation of maven. Editing PATH variable to add maven binaries.

Once, both Java and Maven are successfully installed LAC could be built. In this sense, being in the path of the previously downloaded code, maven has to be run to download all the dependencies of LAC as follows.

```
$ mvn install
```

When all the dependencies have been downloaded, JAR could be built. Maven is already configured to include all the dependencies on the JAR, in that way LAC is totally self-contained. In order to build the JAR of LAC, the following command has to be run. It will create a JAR file at $target/lac - 0.2.0.jar$

```
$ mvn package
```

No additional installation is required, as the JAR is totally self-contained. JAR could also be moved between computers without any compatibility issue.

2.3 Installing LAC

LAC is built as an executable JAR. It could be obtained from the releases page at Github, or it could be built as it was depicted in Section 2.2. In both cases, no installation is required, JAR could be run directly without an installation step. To run LAC, the following command has to be executed, where $config.yml$ is the configuration file of the algorithm/algorithms to run (See Section 4.3 for more details on configuration file).

```
$ java -jar lac-0.2.0.jar config.yml
```

Chapter 3

Algorithms included in LAC

The goal of this section is two-fold. First, a brief description of each algorithm is depicted. Original work is also cited to those who need a further description of each algorithm. Second, a description of each parameter is also given.

3.1 CBA

The very first algorithm for AC proposed by *Liu et al.* [17]. It obtains an associative classifier by means of a two-part process. First, all the rules whose confidence and support are above a user-specified threshold. To obtain those rules, Apriori is used [3]. It is also the first algorithm for association rule mining, whereas for association rule mining the target of discovery is not pre-determined, since no class exists, it is adapted to obtain only a subset of the possible rules. That rules are known as class association rules. Second, once all the class association rules are extracted, CBA continues sorting those rules and removing those which are not covering enough instances of the dataset. In this sense, all the redundancy is removed. Additionally, it also includes a default class for those cases which are not covered by any rule. Default class is obtained as the majority class for the instances not covered while training.

The parameters for this algorithm are the following.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. Authors recommended to use a value of 1%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 50%.

3.2 CBA2

It emerges as evolution of CBA. It was proposed by the very same authors [18]. Unlike its previous version, it includes multiple minimum class supports, that is, a different minimum support per class. In this sense, imbalance among the frequency of occurrence of the classes do not affect to the accuracy of the classifier. Whereas in the previous approach (CBA) rule with minority classes may be totally ignored because they do not achieve enough frequency of occurrence, in this approach this problem is solved. Each support of rule is evaluated with regard to the frequency of its class, enabling to obtain rules for each class without mattering how frequent this class is.

The parameters for this algorithm are shown below.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. For each class c_i, a different minimum class support is assigned. The user only gives a total minimum support, denoted by t_minsup, which is distributed to each class according to their class distributions ($freqDist$) as follows: $minsup_i = t\_minsup \times freqDist(c_i)$. The formula gives frequent classes higher minsups and infrequent classes lower minsups. This ensures that the classifier will generate sufficient rules for infrequent classes and will not produce too many overfitting rules for frequent classes. Authors recommended to use a value of 1%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 50%.

3.3 CMAR

Once of the pioneering algorithms which incorporates prediction of unseen examples using multiple rules, and not only one. Authors proposed this algorithm to solve the problem of classification on those datasets where the classification is based on only single high-confidence rule [15]. As happens previously, this algorithm obtains rules in a two-step process. First, it obtains rules using an algorithm for association rule mining. In concrete, CMAR uses FP-Growth which enables to obtain rules without candidate generation thanks to a data structure that speeds up the process [12]. Second, once all the rule have been obtained, they are sorted and filtered using database coverage. Prediction of unseen examples makes use of weighted X^2 analysis using multiple strong association rules [15].

The parameters for this algorithm are shown below.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. Authors recommended to use a value of 1%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 50%.

- Delta. Minimum number of times that an instance needs to be covered to consider this example as totally covered. Authors recommended a value of 4.0.85

3.4 CPAR

CPAR follows a two-step process to obtain the final classifier. First, it obtains all the rules by means of an adaptation of First Order Inductive Learner (FOIL) [24] algorithm in rule generation [29]. The resulting rules are merged to form the classifier rule set. Rules are pruned considering Laplace accuracy measure. CPAR uses the best k rules of each group to predict the class label of each new tuple.

The CPAR algorithm is the first AC technique that used Laplace Accuracy to assign the class to the test cases during prediction. Once all rules are found and ranked, and a test case (t) is about to be predicted, CPAR iterates over the rule set and marks all rules in the classifier that may cover t. If more than one rule is applicable to t, CPAR divides them into groups according to the classes, and calculates the average expected accuracy for each group. Finally, it assigns to t the class with the largest average expected accuracy value. The expected accuracy for each a rule (R) is obtained as follows:

$Laplace(R) = \frac{support(R)+1}{(support\_antecedent(R)+p)}$

Where p is the number of classes in the training dataset. $support\_antecedent(R)$ is the number of training cases matching R antecedent. $support(R)$ is the number of training cases covered by R that belong to class c.

The parameters for this algorithm are the following.

- Gain. When using FOIL, gain is used to select the best literals at that moment. It measures the information gained from adding this literal to the rule. It is defined as follows for a literal p.

  $gain(p) = |P*|(log\frac{|P^*|}{|P^*|+|N^*|} - log\frac{|P|}{|P|+|N|})$

  Where $|P|$ are the positive examples in the dataset, $|N|$ are the negatives one. After literal p is added to a rule, there are $|P*|$ positive and $|N*|$ negative examples satisfying the new rule's body. Authors recommended to use a value of 0.7.

- Number of rules combining for every example in rule generation phase. Authors recommended to use a value of 0.05. Authors claim that using that small threshold guarantees that many rules are used to generate new candidates, obtaining almost an exhaustive search algorithm.

- Decay factor. When an example is covered by a rule, it is not directly removed but its weight is decreased applying a decay factor (α). The higher the value, the larger the number of rules extracted by example. Authors recommended to use a value of 2/3.

- Number of rules to use in prediction. When a new example is considered, multiple rules are used to perform the prediction. This number of rules could be configured using this parameter, known as k. Authors recommended to use a value of 5.

3.5 MAC

MAC also follows a two-step process to obtain the final classifier [1]. It utilises vertical mining approach [30] for finding the knowledge from data sets whereas the majority of AC algorithms employ level-wise search in discovering the rules (Apriori approach [3]). The main advantage of using vertical mining is that the data set is scanned only once, and then simple intersections among the TIDs of ruleitems of size one are required to derive the remaining ruleitems which are the input for rule production step [30]. Secondly, the classifier is built using the previously obtained rules. In this sense, rules are sorted, and MAC iterates over the set of discovered rules (top-down fashion) and marks the first rule that matches the training case as a classifier rule. The same process is repeated until all training cases are utilised. This methodology of constructing the classifier does not consider the similarity between the candidate rule class and that of the training case during the selection of the classifier rules. Authors claim that reduce overfitting of the resulting classifier as well as its size.

The parameters for this algorithm are the following.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. Authors recommended to use a value of 2%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 40%.

3.6 ACCF

ACCF makes use of a two-step process to obtain the final classifier [16]. ACCF generates a much smaller set of high-quality predictive rules from the datasets thanks to obtaining closed frequent itemsets [22]. An itemset is closed in a dataset if there are no superset that has the same support count as this original itemset. In this sense, ACCF enables to obtain a smaller number of rules without losing accuracy and speeding-up the process of rule discovery. Secondly, rules are sorted following the same criteria as in CBA with a small difference. Whereas CBA considers that a rule $X \rightarrow Y$ covers an example (t) iff $X \subset t$, ACCF relax this condition for that cases where no rule covers completely an example. In that cases, ACCF selects the first rule which contains exactly one item in t. In that way, authors claim that no default class is required.

The parameters for this algorithm are the following.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. Authors recommend to use a value of 1%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 50%.

3.7 L3

L3 also uses a two-step process to obtain the final classifier [5]. First, it obtains rules whose confidence and support is above a user-specified threshold. It makes use of an adaptation of FP-Growth [12] to obtain rules, adapted to incorporate multiple support threshold to solve the imbalance class problem. Then, a classifier is formed using the previously obtained rules. In this last step is where there are very much novelty in this algorithm. It proposes to perform a lazy pruning, the idea behind lazy pruning is to discard from the classifier only the rules that do not correctly classify any training case, i.e., the rules that only negatively contribute to the classification of training cases. To this end, after rule sorting, the training cases are covered to detect "harmful" rules. After having discarded "harmful" rules, the remaining rules are divided in two groups: 1) used rules, which have already correctly classified at least one training case; 2) spare rules, which have not been used during the training phase, but may become useful later.

The parameters for this algorithm are the following.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. For each class c_i, a different minimum class support is assigned. The user only gives a total minimum support, denoted by t_minsup, which is distributed to each class according to their class distributions as follows: $minsup_i = t\_minsup \times freqDist(c_i)$ The formula gives frequent classes higher minsups and infrequent classes lower minsups. This ensures that we will generate sufficient rules for infrequent classes and will not produce too many overfitting rules for frequent classes. Authors did not use this parameter in their experimental studies, but they obtained an average support of 1%. They claimed that this value may be a good starting point.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors did not fixed this parameter in its experimental studies, thus, they used as 0%.

3.8 ACN

This algorithm not only enables to obtain classifier with positives rules, but also with negative ones [14]. It is based on a two-step process. The goal for the first step is to obtain both positive and negative rules. While the second phase aims at sorting and post-processing those rule to obtain an accurate classifier. In this last phase, ACN also calculates pearson's correlation coefficient for each rule and prunes a rule if its correlation measure is below a user-specified threshold.

The parameters for this algorithm are the following.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. Authors recommended to use a value of 1%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e.

$confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 50%.

- Pearson correlation coefficient. It determines the level of correlation between antecedent and consequent. In a formal way, the Pearson correlation coefficient for a rule $R = X \rightarrow Y$ is calculated as $pearson(R) = \frac{support(R) - support(X) * support(Y)}{\sqrt{support(X) * support(Y) * support(\overline{X}) * support(\overline{Y})}}$. Authors recommended a threshold of 0.2.

- Accuracy. This metric is used when building the classifier, negative rules are not added directly to classifier but they have to surpass a threshold of accuracy in training dataset. For a rule R it could be defined as $accuracy(R) = \frac{support(R)}{N}$, where N is the number of instances in dataset. Authors recommended to use a maximum value of 55%.

3.9 ADT

This algorithm aims at obtaining rules whose confidence is higher than a user-specified threshold. Thus, this algorithm does not consider support as all the previous ones but only confidence. As for this algorithm mining confident rules does not require a minimum support, the classic support-based algorithms [3, 12] cannot be used. They proposed to exploit a confidence-based pruning to prune unpromising rules as early as possible. Furthermore, the lattice of rules are converted to a tree called ADT [27] where general rules are at higher level and specific rules are at low levels. In that way the process of pruning could be done more efficiently.

The parameters for this algorithm are the following.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 50%.

- Merit. When pruning rules, ADT aims at removing any remaining rule that covers many cases incorrectly. In this regard, the metric *merit* is defined to find those rules which does not cover many cases correctly. Thus, authors defined merit of a rule $(R = X \rightarrow Y)$ as $merit(R) = \frac{support(X) - support(X \rightarrow \neg Y)}{support(X)}$. Authors recommended to use a maximum value of 10%.

3.10 ACAC

ACAC obtains the classifier directly, and it does not require a two step process as many other algorithm for AC [13]. Whereas almost all the algorithms in AC are based on the traditional framework of support and confidence, this method adds a new metric, known as all-confidence [20]. It represents the minimum confidence of all association rules extracted from an itemset. Thanks to this measure, authors claim to reduce the number of generated rules speeding up the process. This reduction is so high, that authors claim that its algorithm does not have to do a post-processing step since all the generated rules are enough interesting to consider in the classifier.

The parameters for this algorithm are the following.

- Support. It determines the frequency of occurrence. The support of an association rule $X \rightarrow Y$ is defined as the number of transactions from $\mathcal{T}$ that satisfies both X and Y, i.e. $\{\forall t_j \in \mathcal{T}, X \subset t_j \wedge Y \subset t_j : t_j \subseteq \mathcal{I}\}$. The higher the value, the less rules are found, since it only obtains those which are more frequent. Authors recommended to use a value of 2%.

- Confidence. It determines the strength of implication of the rule, so the higher its value, the more accurate the rule is. In a formal way, the confidence measure is defined as the proportion of transactions that satisfy both the antecedent X and consequent Y among those transactions that contain only the antecedent X, i.e. $confidence(X \rightarrow Y) = support(X \rightarrow Y)/support(X)$. Authors recommended to use a maximum value of 100%.

- All-confidence. It represents the minimum confidence of all association rules extracted from an itemset [20]. Let R be a rule with the form $X \rightarrow Y$, where a data object is said to match an itemset $X = \{a_{i1}, ..., a_{ik}\}$ iff for $(a \leq j \leq k)$, the object has value a_{ij} in $Attribute_{ij}$. Thus, all-confidence could be defined as $allconf(R) = \frac{support(R)}{max(support(a_{i_1}),...,support(a_{i_k}))}$. Authors recommended to use a maximum value of 50%.

Chapter 4

LAC Software for users

The goal of this section is five-fold. First, all the different kind of input files will be described. Second, the different reports which LAC is able to create will be explained. Third, the configuration format will be detailed and a couple of examples will be provided to facilitate its use. Fourth, an example on how to run an existing algorithm will be provided. Finally, the automation framework for experimental studies will be fully explained.

4.1 Input of datasets

LAC supports multiple input datasets. It has been developed in this sense to facilitate the integration with existing tools that imposes a specific format. At current version, LAC is able to work with CSV [25], ARFF [10] and KEEL [26] format without any preprocessing step to convert among formats. No configuration is required to use an input or another, but LAC is able to automatically detect the kind of used dataset (by means of the file extension). Next, each format is briefly described and different sources for each kind of input format are provided.

- CSV format [25]. A Comma-Separated Values (CSV) file is a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. At the moment, LAC supposes that the very first row is the header of the dataset and it is not considered as data. Types of attributes are inferred using all the values for each attribute. Class is obtained using the header, or in cases where the attribute class is not represented with this name, the last column is used as class. File extension for this format should be *.csv*. Next, an example is provided.

```
outlook, humidity, windy, temperature, play
sunny, high, low, 25, yes
overcast, low, low, 14, no
rainy, mild, high, 0, no
```

In this example as there are not any attribute with the name of class, the last column, i.e., *play* will be used as class. Whereas this kind of format is not so much used in the research community, it is one of the most used while approaching real-world problems because it is pretty easy to generate. Datasets for this format could be downloaded from `https://www.mldata.io/datasets/`. Additionally, LAC repository also provides a small example for this kind of file at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/examples/dataset.csv`.

- ARFF format [10]. An ARFF file is a text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first section is the header information, which is followed by the data section. The header of ARFF file contains the name of the relation, that is, the name of dataset, a list of attributes, and their respective types. Each line of this section must start with the '@' character. Header section is finished after the line @*data*, where all the data will be found. Each line represents a different instance, and each attribute's value is separated by comma. This format is well-known thanks to Weka [10] tool. File extension for this format should be $.arff$. Next, an example is provided.

```
@RELATION weather.tennis
@ATTRIBUTE outlook {sunny, overcast, rainy}
@ATTRIBUTE humidity {high, low, middle}
@ATTRIBUTE windy {low, high}
@ATTRIBUTE temperature NUMERIC
@ATTRIBUTE play {yes, no}
@DATA
sunny, high, low, 25, yes
overcast, low, low, 14, no
rainy, mild, high, 0, no
```

One of the best source for this kind of dataset, is UCI repository `https://archive.ics.uci.edu/ml/datasets.php`. This repository shares more than 400 datasets, many of them are well-known in the research community and many AC algorithms have been executed on that obtaining very good results. Additionally, LAC repository also provides a small example for this kind of file at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/examples/dataset.arff`.

- KEEL format [26]. It is a text file that describes a list of instances, it is based on ARFF but it also has added a couple of modifications. First, it has changed from *numeric* types to *integer* or *real*. It also adds the intervals for those kind of attributes. Finally, it also requires to specify which attributes are part of the inputs and which are part of the output. As well as ARFF, files have two distinct sections. The first section describes the header information and the second part shows the data. Likewise, each line of this section must start with the '@' character. Header section is finished after the line @*data*, where all the data is found. Each line represents a different instance, and each attribute's value is separated by comma. File extension for this format should be *.dat*. Next, an example is provided.

```
@RELATION weather.tennis
@ATTRIBUTE outlook {sunny, overcast, rainy}
@ATTRIBUTE humidity {high, low, middle}
@ATTRIBUTE windy {low, high}
@ATTRIBUTE temperature integer [0,25]
@ATTRIBUTE play {yes, no}
@INPUTS outlook, humidity, windy, temperature
@OUTPUTS play
@DATA
sunny, high, low, 25, yes
overcast, low, low, 14, no
rainy, mild, high, 0, no
```

The best repository for this kind of dataset, is KEEL repository `http://keel.es/datasets.php`. It has almost 100 datasets for the classification task. Additionally, LAC repository also provides a small example for this kind of file at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/examples/dataset.dat`.

4.2 Reports of algorithms

When an algorithm is run, LAC outputs some very basic information on *STDOUT*. This output cannot be configured, and is prefixed by design. It contains the following basic information.

- Name of the algorithm.
- Runtime, both in training and test phases. Time is measured in milliseconds.
- Number of rules used in the classifier.
- Accuracy for both training and test.

Each execution is represented as follows, where all this basic information is provided.

```
***********************************************************************************************
Algorithm: CBA
Dataset: weather.tennis
Runtime: 1510 ms (Building classifier 1005 ms; Test phase 505 ms)
Number of rules: 5
Training accuracy: 1.0
Test accuracy: 1.0
***********************************************************************************************
```

Although this output provides some basic information, it could not be enough for complex experimental studies. In this regard, LAC also enables to output more advanced reports and persists on disk to be checked whenever is required. By means of the configuration file, reports could be turn on or off, as well as the types of reports to be used. By default, reports are not mandatory and they may or may not be used. Section 4.3 explains in depth how to use the configuration file to specify reports. The three available reports provided by LAC are the following.

1. **KlassesReport**. It stores both the prediction class and the real class for both datasets (train and test). It generates two different files, one for training and another one for test. Generated files are of type CSV with two columns, and one header line at the beginning. The first column has the real class and the second one has the prediction done by the classifier. Additionally, it should be pointed out that the name of the class contains *K* instead of *C* since *class* is a reserved word in Java. LAC source code follows the convection of calling *klass* to the output variable. Next an example is provided for a dataset with two different classes (no and yes).

   ```
   realKlass, predictedKlass
   no, no
   yes, no
   yes, yes
   ```

2. **ClassifierReport**. It stores all the rules forming the classifier. It is a text plain file, where each line has one rule. Both antecedent and consequent are separated by the characters =>.

   ```
   outlook=sunny temperature=hot => no
   outlook=sunny temperature=hot humidity=high => no
   outlook=sunny temperature=mild humidity=high => no
   outlook=normal => yes
   outlook=FALSE => yes
   outlook=sunny => no
   outlook=high => no
   ```

3. **MetricsReport**. It stores all the available metrics (*accuracy, cohen's kappa, recall, precision, f-measure (macro and micro)*, number of rules and average number of attributes per rule) and the confusion matrix. The runtime of the different phases are also persisted. Generated files are of type CSV, where the first column is the name of the metric and second column is the value.

```
Name metric, Value
Accuracy, 0.7697749196141479
Kappa, 0.5705294763867973
Recall, 0.5606825062334041
Precision, 0.832909604519774
F-measure(micro), 0.8006688963210703
F-measure(macro), 0.6068756056725807
Number rules, 47
Average number attributes, 4.148936170212766
Runtime training (ms),448
Runtime test (ms),31
```

Nevertheless, each developer could define his own reports and use it on LAC without any problem.

4.3 Configuration

Configuration is done using YAML [6] (see Appendix A for better description). This format enables to write configurations files in a human-readable format, and being very easy to understand. This file is responsible for selecting the algorithms to be executed, their parameters, datasets and type of reports. YAML files are text plain and typically have a *.yml* extension. Appendix A better describes general types for YAML files. This configuration file in LAC has the following structure.

First of all, configuration file needs to have a root parent called *executions* of array type. Each element for this array will specify one execution, they could be of the same algorithm or dataset, or it could be different, there are not any kind of restriction in this sense. The content of each element for *executions* has to have the following properties:

- *name_algorithm*: it is a mandatory property, of type *String*, containing the name of the algorithm to be executed.

- *configuration*: it is an optional property, of type *Object*, containing the different parameters of configuration for the algorithm. In cases where this parameter is not provided, default configuration will be used, i.e., those provided by the authors. The elements for this Object depends on the algorithm being run, see Section 3 for further details.

- *train*: it is a mandatory property, of type *String*, containing the relative or absolute path for the dataset used in training phase.
- *test*: it is a mandatory property, of type *String*, containing the relative or absolute path for the dataset used in test phase.
- *reports*: it is an optional property, of type *String*, containing the relative or absolute path where all the reports will be saved. In cases where this property is not provided, no report will be generated.
- *reports_type*: it is an optional property, of type Array, containing each type of reports to be saved. Each element of the array must be of type *String*, see Section 4.2 for further details.

Therefore, the typical structure for a LAC configuration file is as follows.

```
executions:
  - name_algorithm: "Name of the algorithm 1"
    configuration: { Property1: value1, property2: value2 }
    train: "name of the dataset to use in training phase"
    test: "name of the dataset to use in test phase"
    reports: "path where the results will be persisted"
    reports_type: ["Type of report 1", "Type of report 2"]
  - name_algorithm: "Name of the algorithm 2"
    configuration: { Property1: value1, property2: value2 }
    train: "name of the dataset to use in training phase"
    test: "name of the dataset to use in test phase"
    reports: "path where the results will be persisted"
    reports_type: ["Type of report 1", "Type of report 2"]
```

Once the schema of configuration file has been described, practical examples are provided. First, an example on how to run CBA [17] algorithm on weather dataset is provided. At the very beginning the root element *executions* is declared. Then, the name of the algorithm is specified using *name_algorithm*. Its configuration is also changed to use $min\_sup = 0.01$ and $min\_conf = 0.9$. Finally, reports are configured by means of both *reports* and *reports_type*. For this last parameter two different reports are used (See Section 4.2 for further details).

```
executions:
  - name_algorithm: "CBA"
    configuration: { min_sup: 0.01, min_conf: 0.9 }
    train: "weather-training.arff"
    test: "weather-test.arff"
    reports: "results/weather.nominal"
    reports_type: ["ClassifierReport", "MetricsReport"]
```

Moreover, LAC also enables to execute multiple algorithms. Next an example on how to run LAC to use three different executions is provided. There are not any differences with regard to the previous examples, but *executions* property instead of having one unique element, it has three elements. No configuration nor reports are extracted in this case, thus only mandatory parameters are specified. In this concrete case, CBA [17] is run on two different datasets (iris and weather) and CPAR [29] is only run on one unique dataset (weather).

```
executions:
  - name_algorithm: "CBA"
    train: "weather-training.arff"
    test: "weather-test.arff"
  - name_algorithm: "CPAR"
    train: "weather-training.arff"
    test: "weather-test.arff"
  - name_algorithm: "CBA"
    train: "iris-training.arff"
    test: "iris-test.arff"
```

At the Github repository, there are two examples. First, a basic configuration file to run CMAR [15] could be found at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/examples/cmar.yml`. Then, a more complex one, where all the available algorithms of LAC are executed, is also at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/examples/all.yml`

4.4 Running an existing algorithm

In this section an example on how to run an existing algorithm will be detailed. In concrete, the used algorithm is known as CMAR [15]. It will be configured to use two personalized parameters (See Section 3.3). *min_sup* will be set to 1%, and *min_conf* will be set to 80%. This algorithm will be run on weather.nominal dataset using the arff format. It could be downloaded from Weka Repository at `https://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html`.

```
executions:
  - name_algorithm: "CMAR"
    configuration:
      min_sup: 0.001
      min_conf: 0.9
    train: "weather-training.arff"
    test: "weather-test.arff"
    reports: "results/weather"
    report_type:
      - "KlassReport"
    - "ClassifierReport"
```

To run this configuration file, supposing that it saved as *config.yml* in current path, LAC will have to be called running the following command.

```
$ java -jar lac-0.2.0.jar config.yml
```

Once the algorithm will have finished its execution, it will show additional information to those shown in report files. For instance, it will show runtime for training and testing phases, separated and aggregated. It will also show the number of used rules in the classifier, and metrics of quality for the solutions. Next the output of this execution is shown.

4.5 Using the framework to automate the experimental study

In this section a typical experimental study for a new proposal, where many comparisons have to be made is explained. Whereas in the previous section a basic example was carried out, in this one it is performed a much more complex one. The goal is not only to prove that LAC enables to do it, but to prove how easy is thanks to its configuration file and design. Next, it has been considered interesting to explain how several tools have tried to solve this problem, and how LAC has solved all of them.

- **Using many files.** This is one of the most typical use cases for existing tools, where many configuration files are used. The goal of each configuration file is to run one algorithm in only one dataset. To better illustrate this case, let suppose an experimental study where 10 algorithms are being compared using 30 datasets. Supposing at least one configuration file per algorithm and per dataset, $10 \times 30 = 300$ files will have to be used. In each one of these files the specific configuration of the algorithms will have to be repeated, thus, if you need to adapt one parameter for one algorithm in all the datasets, at least 30 configuration files will have to be edited. Basically, the duplication of those files are very high hampering the maintainability. Last but not least, to run each one of these files, a different execution will have to be carried out since very few tools enables to pass multiple configuration files and start running those in a sequential fashion. Even less if those configuration files want to be run in a parallel way. In summary, this approach hampers the automation of the experimental studies or it forces to develop external tools to facilitate its automation. Basic computer science principles as DRY (Do not Repeat Yourself) are not respected using this methodology [9].

- **A unique configuration file.** In order to solve the problem of dealing with so many configuration files, some tools have decided to join all these files in only one. Although it could facilitate dealing with changes, since only one big file will have to be changed. It is not a solution when a large experimental study is carried out, since this file could grow too much to be readable. For instance, let suppose previous example of 10 algorithms on 30 datasets, supposing also that 10 lines are required to configure each algorithm in each dataset, file will have $10 \times 10 \times 30 = 3000$ lines. Those large files are very error prone, and the may be avoided as much as possible. Furthermore, again this methodology does not provide a way of solving principles as DRY [9].

- **GUI.** Some tools try to solve the design of the experimental study providing users with graphical interfaces. In the first instance, this approach seems like a solution since researchers do not have to deal with a huge number of configuration of files, but it is not a real solution. If these experimental analysis designed using the GUI cannot be exported to configuration files, they could not be run on servers or clusters where no graphical interface is typically installed. When GUI enables to export, the same problem as dealing with a huge number of files arises.

LAC has been specifically designed with all these problem in minds to address all of them. First of all, configuration file could be split in many files or only use one it is entirely up to the user. Also the problem of DRY [9] is addressed using YML which enables to not repeat common parts, facilitating the maintainability. In this sense, when LAC is executed it could receive from one configuration file as $java$ $-jar$ lac-0.2.0.jar $config1.yml$ up to whatever are required as $java$ $-jar$ lac-0.2.0.jar $config1.yml$ ... $configN.yml$. It also enables to use regular expressions for selecting configuration files when running LAC, for instance let suppose a directory with the following configuration files.

```
config.cba.dataset1.yml
config.cba.dataset2.yml
...
config.cba.datasetN.yml
config.cpa.dataset1.yml
config.cpa.dataset2.yml
...
config.cpa.datasetN.yml
config.cmar.dataset1.yml
config.cmar.dataset2.yml
...
config.cmar.datasetN.yml
```

If only the configuration files of CBA [17] want to be executed, the naive approach would be to write *java -jar lac-0.2.0.jar config.cba.dataset1.yml config.cba.dataset2.yml ... config.cba.databaseN.yml*. But with LAC is much more easy thanks to regular expressions, the previous command would be equivalent to write this short version *java -jar lac-0.2.0.jar config.cba.dataset ∗ .yml*. It will automatically detect which configuration files match this regular expression, and run each one of those.

By default, LAC runs each algorithm sequentially, that is, if a configuration file has 10 executions, it will not run the second execution until first has totally finished. This behavior is the most common among existing tools [10, 26], however, LAC also enables to run experimental studies in a parallel way. It could be personalized to speed up the execution on those servers where the hardware have enough capabilities. To increase the level of parallelism, an environment variable is used. Therefore, in experimental studies where want to be executed using a parallelism of 5, LAC will be run as follows:

```
$ LAC_THREADS=5 java -jar lac-0.2.0.jar config.yml
```

It will create 5 independent threads, and run each algorithm on one of those threads. The parallelism of each algorithm is not changed, since they were designed to be sequentially run, but each independent execution is parallelized. LAC will not finish until all the executions, that is, all the threads have finished their executions. Environment variable has been used to facilitate its configuration. In that way, the number of threads could be changed in each different use of LAC or it could be configured at system level. Using environment variables for this kind of configuration is well-known and documented, in fact, there are a recent outstanding movement standardizing this way of configuration known as *Twelve-Factor* [28].

Finally, the usefulness of using YML is highlighted. When a complex experimental study is being performed it is much more easy to understand why is really important to not repeat configuration. Typical complex experimental studies involve running multiple algorithms on different datasets, where each algorithm typically shares the configuration among datasets. Let suppose an experimental study where 10 algorithms are being compared using 30 datasets. It means that *executions* has $10 \times 30 = 300$ elements, where each one is a different execution. Thus, naive approach would be as follows.

```
executions:
  - name_algorithm: "CBA"
    configuration:
      min_sup: 0.001
      min_conf: 0.9
    train: "dataset1-training.arff"
    test: "dataset1-testing.arff"
  - name_algorithm: "CBA"
```

```
    configuration:
      min_sup: 0.001
      min_conf: 0.9
    train: "dataset2-training.arff"
    test: "dataset2-testing.arff"
    ...
  - name_algorithm: "CMAR"
    configuration:
      min_conf: 0.9
      delta: 4
    train: "dataset1-training.arff"
    test: "dataset1-testing.arff"
  - name_algorithm: "CMAR"
    configuration:
      min_conf: 0.9
      delta: 4
    train: "dataset2-training.arff"
    test: "dataset2-testing.arff"
    ...
```

In this example, two things are repeated. Firstly, name of the datasets are duplicated in each algorithm, that is, as 10 algorithms are being run the same name of the dataset is repeated in 10 different places of the configuration file. If one of those files is changed of path, it requires changing the same path in 10 different places. Secondly, the configuration for each algorithm is also repeated, thus, if a change has to be made 30 different place have to be changed hampering the maintainability of this file. Traditional computer science principles as DRY [9] are totally being violated. However, this problem could be easily solved thanks to using inheritance of YML. It enables to not repeat ourself many times in one configuration file, facilitating whatever change is required. Inheritance in YML work as follows.

```
.dataset1: &dataset1
  train: "dataset1-training.arff"
  test: "dataset1-testing.arff"

.dataset2: &dataset2
  train: "dataset2-training.arff"
  test: "dataset2-testing.arff"

...

.config_cba: &config_cba
  configuration:
    min_sup: 0.001
    min_conf: 0.9

.config_cmar: &config_cmar
  configuration:
    min_conf: 0.9
    delta: 4
```

```
...

executions:
  - name_algorithm: "CBA"
    <<: *config_cba
    <<: *dataset1
  - name_algorithm: "CBA"
    <<: *config_cba
    <<: *dataset2
    ...
  - name_algorithm: "CMAR"
    <<: *config_cmar
    <<: *dataset1
  - name_algorithm: "CMAR"
    <<: *config_cmar
    <<: *dataset2
    ...
```

First, an alias for each dataset is defined as *datasetX* where both train and test file are declared. This alias is included in each execution using the syntax <<: $*datasetX$. In that way, if *datasetX* were changed of path, only the alias would have to be changed in one unique place. Second, an alias for each configuration's algorithm is also declared with the form $config\_nameAlgorithm$ this configuration is included in each use of *nameAlgorithm*. In this sense, if one parameter of *nameAlgorithm* were changed, it would only have to be changed in one unique place (where the alias was defined).

Chapter 5

LAC Software for developers

The goal of this section is to better describe how easily LAC could be extended adding new features. First, an example on how to add a new input is described. Second, an example on how to add a new algorithm is depicted. Ultimately, a new type of report is also added. This section is mainly aimed at developers, thus, non-developer users may skip this whole section.

5.1 Structure of LAC packages/classes

LAC is formed by a couple packages, where each one has its own responsibility. Figure 5.1 shows the class diagram for LAC.

- *lac.algorithms* includes the base classes for all the algorithms, and one package for each algorithm. Each algorithm is contained in one package with the same name as the algorithm. For example, considering the CBA [17] algorithm, the name of the package must be *lac.algorithms.cba*. It also contains three important classes: 1) *Rule* used to represent each obtained rule; 2) *Classifier*, is the parent class representing a classifier, it contains logic for both to save the set of rules and to predict unseen examples by means of the previous rules, and each algorithm should implement it; 3) *Algorithm*, is the parent class representing an algorithm, each algorithm should implement it and it contains the logic to transform from a training dataset to a set of rules.
- *lac.data* includes several classes to represent the data, and one class for each format of input file. *ArrfDataset* enables to run LAC on Arff format, *DatDataset* contains the logic for the KEEL format, and in the same way *CsvDataset* contains the logic for the CSV format. *Attribute* contains some basic logic to work with the attributes

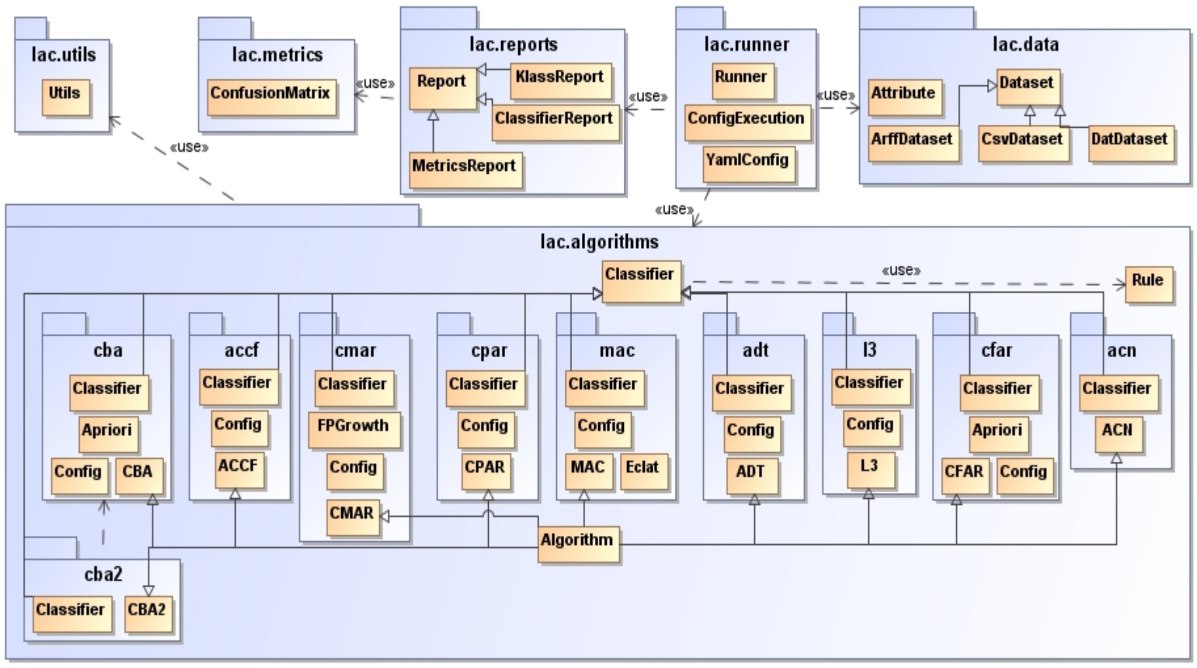


Figure 5.1: Class diagram for LAC.

contained into the dataset, for instance to determine the type or the possible values. Finally, *Dataset* is the base class from which inherits each type of dataset.

- *lac.metrics* includes all the metrics to quantify the quality of the solutions. The following measures are supported: *accuracy, cohen's kappa, recall, precision, f-measure*, number of rules and average number of attributes per rule. *ConfusionMatrix* implements an efficient confusion matrix to calculate all the aforementioned measures.
- *lac.reports* provides reports to show different results of the algorithms. *MetricsReport* calculates all the possible metric supported by LAC. *KlassReport* shows both the predicted and real class in training and test phases. Finally, *ClassifierReport* enables to obtain all the rules forming the final classifier.
- *lac.runner* provides a framework to automate and parallelize the experimental studies. *YamlConfig* contains the logic to read configuration files in YML format. *ConfigExecution* is used to represent each configured execution. *Runner* runs each *ConfigExecution* in a parallel or sequential way.
- *lac.utils* includes some extra and useful methods to work with rule sets.

LAC also contains a complete suite of tests to check that all the features are correctly working. This suite of tests is at *src/tests* and they have been written using both jUnit and mockito. These tests could be run using maven as *mvntest*.

5.2 Adding a new input format

LAC is ready to be extended to add new kinds of input formats. The steps to create a new format are as follows.

1. First of all, a new class file has to be created in *lac.data*. The filename is important and it should follow the name convention: *extensionDataset.java*, where *extension* should be replaced with the extension of the file being integrated. For instance, to add support for *.arff* format, the class *ArffDataset* has to be created. In the same way, to add support for *.csv* format, the class *CsvDataset* has to be created. New class file should extend from *lac.data.Dataset*.

2. The only requirement is to implement a constructor which receives a parameter of type *String*. This parameter contains the path of the file to be read. No other method is required to be implemented.

3. The previously created constructor should have two-goals. First, it should set the metadata information for the dataset. In this sense, it should call the method *addNominalAttribute* or *addNumericAttribute* for each attribute contained in the dataset. The class of the dataset is not considered as an attribute, and metadata of this should be set up calling the method *addKlass*. Second, for each instance contained in the dataset, *addInstance* should be called.

In order to better illustrate this process, a practical step-by-step is provided. In this example, the goal is to be able to read *.arff* format (Section 4.1 provides a full example for this kind of input format as well as the specifications regarding schema).

1. The file *ArffDataset.java* is created in the package *lac.data*, extending from *lac.data.Dataset* as follows.

```java
package lac.data;

public class ArffDataset extends Dataset {
}
```

2. ArffDataset should implement a constructor receiving only a unique parameter. It should be of type *String*.

```java
package lac.data;

public class ArffDataset extends Dataset {
    public ArffDataset(String path) {
    }
}
```

3. In this step, file path received as parameter is opened and read. Both metadata and instances have to be saved. The first step in this constructor is to call parent (See Line 2). After that, file is opened and read (Lines 4-8). In this specific format, for each line of file three options arise. First, line could be empty or a comment, in these cases they are discarded (See Lines 13-14). Second, if line starts with the character '@' it contains metadata information, so a new function has been declared to manage this kind of lines (See Line 16-17). Third, line is a valid instance which will be processed in another function called *processData*.

```
 1 public ArffDataset(String path) throws Exception {
 2     super();
 3 
 4     try {
 5         FileReader fileReader = new FileReader(path);
 6         BufferedReader bufferedReader = new BufferedReader(fileReader);
 7 
 8         String line;
 9         while ((line = bufferedReader.readLine()) != null) {
10             // Remove empty spaces
11             line = line.trim();
12 
13             // Ignore empty lines
14             // Ignore lines with comments
15             if (line.isEmpty() || this.startsWith("#") || line.startsWith("%")) {
16                 continue;
17                 // Metadata information
18             } else if (line.startsWith("@")) {
19                 this.proccessMetadata(line);
20             } else {
21                 // Check if no class was set, the last attribute will be used
22                 if (this.getKlass() == null) {
23                     Attribute attrKlass = this.getAttribute(this.getNumberAttributes() - 1);
24                     this.attributes.remove(this.getNumberAttributes() - 1);
25                     this.indexes.remove(this.getNumberAttributes());
26                     this.addKlass(attrKlass.getValues());
27                 }
28                 this.proccessData(line);
29             }
30         }
31 
32         bufferedReader.close();
33     } catch (Exception e) {
34         System.err.println("File " + path + " cannot be found.");
35         throw e;
36     }
37 }
```

Continuing this example, the function *processMetadata* is entirely defined. As it has been previously stated, the goal of this function is to process those lines which are part of metadata information. In this sense, two kind of lines could be found. First lines containing the name of the dataset (See Line 2-3) or lines containing attribute or class information (See Line 4-6). For the first case, the private field *name* will be set, and the last case will be delegated to the function *processAttribute* defined in parent class *lac.data.Dataset*.

```java
1  private void proccessMetadata(String line) {
2      if (line.toLowerCase().startsWith("@relation")) {
3          this.name = line.replace("@relation ", "");
4      } else if (line.toLowerCase().startsWith("@attribute")) {
5          this.processAttribute(line);
6      }
7  }
```

To continue this example, the function *processAttribute* is better described. The goal of this function is to extract metadata information for attributes and the class. First, line is split by spaces in order to detect the kind (Lines 3-9). As nominal attributes has the format *@attribute name_attribute* $\{value_1, ..., value_i\}$, the character '{' is used to detect whether the attribute is nominal or not. In affirmative case, the value is extracted and saved calling *addNominalAttribute*. On the contrary, the attribute is numeric so it is saved calling *addNumericAttribute*.

```java
1  private void processAttribute(String line) {
2      Pattern p = Pattern.compile("@attribute .*\\{(.*)\\}");
3      Matcher m = p.matcher(line.toLowerCase());
4
5      String[] splitted = line.replaceAll(" +", " ").split(" ");
6      String nameAttribute = splitted[1];
7
8      // Nominal attributes in weka starts with {, to list all the possible values
9      if (m.matches()) {
10         String[] values = m.group(1).replaceAll(" *", "").split(",");
11
12         if (nameAttribute.contains(KLASS)) {
13             this.addKlass(values);
14         } else {
15             this.addNominalAttribute(nameAttribute, values);
16         }
17     } else {
18         this.addNumericAttribute(nameAttribute);
19     }
20 }
```

The complete class created for this example is available at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/developing_new_format/`.

5.3 Developing a new algorithm

LAC is ready to be extended to add new algorithms. The steps to create a new algorithm are as follows.

1. First, a new package has to be created in *lac.algorithms*. The name of the package should be the same as the name of the algorithm. For instance, let suppose that CBA [17] algorithm is being implemented, the package's name should be *lac.algorithms.cba*. As it was expected package's name must be in lowercase.

2. In the created package, a new class has to be generated. The class's name should be the same as the name of the algorithm, that is, let suppose again that CBA [17] is being implemented, class's name should be CBA and extend from $lac.algorithms.Algorithm$.

3. Previous created class has to implement at least two different methods. First, a constructor receiving an instance of $lac.configs.Config$. Second, an method called $train$ returning a $Classifier$ and receiving as argument a $Dataset$. The goal for this last method should be to create an instance of classifier for this kind of algorithm, using the dataset passed as argument as training set.

4. Returning to the created package, a new class has to be generated. The class's name is recommended to be called $Config$, and its aim should be to store all the possible configuration parameters for this algorithm. It should extend from $lac.algorithms.Config$.

5. Returning to the created package, a new class has to be generated. The class's name is recommended to be called $Classifier$, and its aim should be to store all the obtained rules and to have all the logic required to predict unseen examples. It should extend from $lac.algorithms.Classifier$.

6. $Classifier$ class has to implement two methods. First, a constructor should be implemented, its goal is to initialize the state calling $super$. Second, a method called *predict* receiving an array of *short* and returning a *short* may be implemented. The aim of this method is to predict the example received as argument by means of the previously extracted rules. It should be highlighted that LAC stores internally each attributes' value as *short*, in this way much less memory is used. This representation is totally internal, and is managed automatically by *lac.data.Dataset*. Of course, this representation is also completely transparent for end-user and they do not need to preprocess nor do any kind of process. This technique of saving instances as

short and not by the original types (*string* in almost all the cases) is well-known and it is being used by many current tools for both AC and ARM [8, 26].

It should be bored in mind that there were not any kind of registration to make this algorithm works, but LAC is automatically able to detect if these conventions are followed. In this way, it is much more easy to add algorithms. In this sense, developers could completely be focused on his own algorithms and being centered on writing only the core of his algorithms. In order to better illustrate these steps, a practical example is described next. In this example, a fictional algorithm is implemented. Its goal is to obtain one unique rule where the antecedent is empty, and the consequent is the majority/minority class. This kind of rule, where the antecedent is empty and the consequent is the majority class is used by many algorithms in AC to represent the default class [15, 17, 18]. To configure if majority or minority class has to be obtained, a configuration parameter is used (*majority* of type *Boolean*). This fictional algorithm will be named MC ($Majority/MinorityClass$).

1. First of all, the package has to be created. As the name of the algorithm is MC, the package *lac.algorithms.mc* is created.

2. A new class with the name of $MC.java$ is created into *lac.algorithms.mc* extending from *lac.algorithms.Algorithm* as follows.

```java
package lac.algorithms.mc;

import lac.algorithms.Algorithm;

public class MC extends Algorithm {
}
```

3. Next, two methods have to be added as follows.

   - First, a public constructor is created where its goal is to receive the configuration for this algorithm. It is saved on an instance variable called $config$ of type *lac.algorithms.Config* defined in *lac.algorithms.Algorithm* (See Line 7).

```java
1 package lac.algorithms.mc;
2
3 import lac.algorithms.Algorithm;
4
5 public class MC extends Algorithm {
6  public MC(Config config) {
7   this.config = config;
8  }
9 }
```

- Second, a method called *train* should be implemented returning a *lac.algorithms.Classifier* and receiving as argument a *Dataset*. The goal for this last method is to obtain the majority or minority class in function of the user-specified parameter. First, frequency of each possible class is calculated in function *getFrequencyByKlass* returning a hash where the keys are the class and the value is the frequency of occurrence (Lines 25-40). Then, configuration is checked to know whether majority or minority class has to be extracted (Line 11). At the end, the classifier is returned with the unique created rule (Line 22).

```java
package lac.algorithms.mc;

// imports are obviated by space limitations

public class MC extends Algorithm {
   (...)

  public Classifier train(Dataset training) throws Exception {
     HashMap<Short, Long> frequencyByKlass = this.getFrequencyByKlass(training);

     short klass;
     // Generate rule with minority or majority class
     if (((Config) this.config).getMajority()) {
        klass = Collections.max(frequencyByKlass.entrySet(),
                 (e1, e2) -> e1.getValue().compareTo(e2.getValue())
              ).getKey();
     } else {
        // Get this klass with the smallest frequency of occurrence
        klass = Collections.min(frequencyByKlass.entrySet(),
                 (e1, e2) -> e1.getValue().compareTo(e2.getValue())
              ).getKey();
     }

     return new Classifier(new Rule(new short[]{}, klass));
  }

  private HashMap<Short, Long> getFrequencyByKlass(Dataset dataset) {
     HashMap<Short, Long> frequencyByKlass = new HashMap<Short, Long>();

     for (int i = 0; i < dataset.size(); i++) {
        // Get the klass for the instance in position i
        short klass = dataset.getKlassInstance(i);

        Long count = frequencyByKlass.get(klass);
        if (count == null) {
           frequencyByKlass.put(klass, 1L);
        } else {
           frequencyByKlass.put(klass, count + 1);
        }
     }
     return frequencyByKlass;
  }
}
```

4. A new class has to be created to store the configuration of this algorithm. The name's class is *Config* extending from *lac.algorithms.Config*. Furthermore, it has to have one instance variable per parameter. As in this particular case only one parameter exists, only one is declared. This parameter is of type *Boolean* and has a default value of *true* (Line 4). Additionally, a setter and getter have also to be declared (Lines 6-12). It should be pointed that no readers of YML, or any kind of reader, have to be declared. LAC is able to automatically call this class and set its

values thanks to the aforementioned setter.

```java
package lac.algorithms.mc;

public class Config extends lac.algorithms.Config {
    private Boolean majority = true;

    public void setMajority(Boolean majority) {
        this.majority = majority;
    }

    public Boolean getMajority() {
        return this.majority;
    }
}
```

5. Returning to the created package, a new class has to be generated. The class's name is *Classifier* and it must extend from *lac.algorithms.Classifier*.

```java
package lac.algorithms.mc;

public class Classifier extends lac.algorithms.Classifier {
}
```

6. The goal of previous created class is to predict unseen data by means of the previous extracted rules in Step 4. As this algorithm only contains one *Rule*, constructor only receives one unique rule (Line 6). This rule is saved in the list of rules forming this classifier (Line 8), this field is declared at the parent level. As the mechanism of prediction of this algorithm is very easy, there is no need to override parent class *predict*. Originally, this function iterates over *this.rules* and check which rule fires first, returning the class of this rule. This file is declared as follows.

```java
 1  package lac.algorithms.mc;
 2
 3  import lac.algorithms.Rule;
 4
 5  public class Classifier extends lac.algorithms.Classifier {
 6      public Classifier(Rule rule) {
 7          super();
 8          this.rules.add(rule);
 9      }
10  }
```

To sum up, adding a new algorithm is pretty straightforward. The above steps have to be followed and LAC will be able to automatically detects new algorithms. All the logic related with reading datasets or configurations is totally abstracted, in this way, developers could focus on what matter most to them, their algorithms. Complete source code for this example is also available at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/developing_new_algorithm/`.

5.4 Adding a new type of report

LAC is ready to be extended adding new types of reports. The steps to create a new type of report are as follows.

1. First of all, a new class file has to be created in *lac.reports*. The filename is recommended to follow the name convention of *descriptiveNameReport* where *descriptiveName* should be replaced by a name which better describe the content of this report. Created class has to extend from *lac.reports.Report*.

2. It should implement a constructor receiving one unique parameter of type *lac.runner.ConfigExecution*. This constructor must call *super*.

3. It should also implement a method called *write* which receives one unique parameter of type *String* containing the path to be used to store the report.

In order to better illustrate these steps, next a practical example how to add a new type of report is much more described. In this example, the type *KlassReport* is added. The goal of this report is to save both the prediction and the real classes for the dataset.

1. The file *KlassReport.java* is created in the package *lac.reports*. This file declares *KlassReport* extending from *Report*.

```java
package lac.reports;

import lac.runner.ConfigExecution;

public class KlassReport extends Report {
}
```

2. *KlassReport* should implement a constructor receiving one unique parameter of type *lac.runner.ConfigExecution* and calling *super* as follows.

```java
package lac.reports;

import lac.runner.ConfigExecution;

public class KlassReport extends Report {
    KlassReport(ConfigExecution config) {
        super(config);
    }
}
```

3. Finally, *KlassReport* should implement the method *write*. As in this concrete case, two different outputs are generated (one for training and another for test), this logic has been moved to another private method called *writeResults*. The goal for this function is to generate both real and predicted class for the dataset passed as parameter and saving into the path. It also important to note that as it was previously highlighted, that both attributes and classes are internally represented in LAC using *short*. In this way, dataset and rules occupy a very small part of memory compared to saving the same information as string. To be transparent with the end-user, only the real value, and not the internal representation, is saved. To transform from internal representation in short to original value, *dataset.getValueByIndex* could be used.

```java
package lac.reports;

(...)

public class KlassReport extends Report {
    (...)

    public void write(String reportPath) {
        writeResults(this.training, reportPath + ".training.csv");
        writeResults(this.test, reportPath + ".test.csv")
    }

    private void writeResults(Dataset dataset, String reportPath) {
        try {
            PrintWriter writer = new PrintWriter(reportPath, "UTF-8");

            // Write header information
            writer.println("realKlass, predictedKlass");

            for (int i = 0; i < dataset.size(); i++) {
                short[] example = dataset.getExample(i);
                short predictedKlassIndex = this.classifier.predict(example);
                short realKlassIndex = dataset.getKlassInstance(i);

                // Classes are transformed from internal representation to original value
                String realKlass = dataset.getValueByIndex(realKlassIndex);
                String predictedKlass = dataset.getValueByIndex(predictedKlassIndex );

                writer.println(realKlass + "," + predictedKlass);
            }
            writer.close();

        } catch (FileNotFoundException | UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

The complete source code is available at `https://github.com/kdis-lab/lac/tree/v0.2.0/doc/developing_new_report/`.

Chapter 6

Reporting bugs

Feel free to open an issue at Github if anything is not working as expected `https://github.com/kdis-lab/lac/issues`. Merge requests are also encouraged, it will be carefully reviewed and merged if everything is all right.

Acknowledgements

This research was supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund, Project TIN-2017-83445-P.

Appendix **A**

YAML format

The goal of this section is to better describe the syntax of YAML. It has been considered interesting to add this appendix to enable readers to get more out of the configuration of LAC.

YAML stands for "YAML Ain't Markup Language" and it is a human friendly data serialization language, providing the same functionalities as some other formats as JSON or XML but being less complex. It has been typically used for configuration files, but it is not limited to only that. YAML files should end in *.yml* whenever possible. It is also case-sensitive, and it does not support tabs but spaces should be used. For further information on the format, specification could be found at `https://yaml.org/spec/1.2/spec.html` and there are also available libraries for different languages at `https://yaml.org`.

Data types

Next each data type are fully described. First, different number types are supported. Integer values are declared as 12, float values are declared as 12.55, even exponential number could also be declared as $1.2e + 34$. Boolean data types are also supported as $true$ and $false$. YAML even includes the possibility of adding comments with lines are prefixed by the character ‘’. String are also supported, they can be wrapped both in single and double quotes. In some cases, they can also be unquoted. Quoted styles are useful when a string starts or end with one or more relevant spaces, because unquoted strings are trimmed on both end when parsing their contents. Quotes are required when the string contains special or reserved characters (:, , , [,], ,, , $*$, , ?, |, $-$, $<$, $>$, $=$, !, %, @, ‘).

Continuing with a more complex data type, array type enables to model a sequence of items. Two different syntaxes exist:

1. Single-line: where all the elements are listed on the same line. Each element is separated by the character ‘,’, and it should be limited by [...]. Next, an example is provided.

```
array: [1, 2, 3, 4]
```

2. Multiple-line: where each element is in a different line. It should be used for large array to maintain readability. Each element should start by the character ‘−’, and being tabulated one level more than the parent. Next, an example is provided.

```
array:
  - 1
  - 2
  - 3
  - 4
```

Continuing with the next data type, mappings are described. It gives the ability to list keys with values. This is very useful for those cases where you are assigning a name or a property to a specific element. As happens with array, two syntaxes are supported.

1. Single-line: where all the set of properties/values are listed on the same line. Each set of key/value pair is written as *key* : *value*, where each different *key* is separated by the character ‘,’, and it should be limited by {...}. Next, an example is provided.

```
object: {key1: 1, key2: 2}
```

2. Multiple-line: where each set of key/value is in a different line. It should be used for large objects to maintain readability. Each set of key/value pair should be in a different line, and being tabulated one level more than the parent. Key is separated from value by means of the character ‘:’. Next, an example is provided.

```
object:
  key1: 1
  key2: 2
```

Bibliography

[1] N. Abdelhamid, A. Ayesh, F. Thabtah, S. Ahmadi, and W. Hadi, “Mac: A multiclass associative classification algorithm,” *Journal of Information & Knowledge Management*, vol. 11, 06 2012.

[2] N. Abdelhamid and F. Thabtah, “Associative classification approaches: Review and comparison,” *Journal of Information & Knowledge Management*, vol. 13, no. 03, p. 1450027, 2014.

[3] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.

[4] M. Z. Asghar, A. Khan, A. Bibi, F. M. Kundi, and H. Ahmad, “Sentence-level emotion detection framework using rule-based classification,” *Cognitive Computation*, vol. 9, no. 6, pp. 868–894, Dec 2017.

[5] E. Baralis and P. Garza, “A lazy approach to pruning classification rules,” in *Proceedings of the 2002 IEEE International Conference on Data Mining*, ser. ICDM ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 35–. [Online]. Available: http://dl.acm.org/citation.cfm?id=844380.844721

[6] C. C. E. Brian Ingerson and O. Ben-Kiki. (2019) Yet another markup language (yaml) 1.0. [Online]. Available: https://yaml.org/spec/history/2001-05-26.html

[7] C. Cortes and V. Vapnik, “Support vector networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.

[8] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng, “Spmf: A java open-source pattern mining library,” *Journal of Machine Learning Research*, vol. 15, pp. 3569–3573, 2014.

[9] A. Frömmgen, D. Stohr, B. Koldehofe, and A. Rizk, "Don't repeat yourself: Seamless execution and analysis of extensive network experiments," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: ACM, 2018, pp. 20–26.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[11] J. Han and M. Kamber, *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2011.

[12] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, Jan 2004.

[13] Z. Huang, Z. Zhou, T. He, and X. Wang, "Acac: Associative classification based on all-confidence," 11 2011, pp. 289–293.

[14] G. Kundu, M. M. Islam, S. Munir, and M. F. Bari, "Acn: An associative classifier with negative rules," in *2008 11th IEEE International Conference on Computational Science and Engineering*, July 2008, pp. 369–375.

[15] W. Li, J. Han, and J. Pei, "Cmar: Accurate and efficient classification based on multiple class-association rules," in *2001 IEEE International Conference on Data Mining(ICDM01)*, 2001, pp. 369–376.

[16] X. Li, D. Qin, and C. Yu, "Accf: Associative classification based on closed frequent itemsets," 11 2008, pp. 380 – 384.

[17] B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rule mining," in *4th International Conference on Knowledge Discovery and Data Mining(KDD98)*, 1998, pp. 80–86.

[18] B. Liu, Y. Ma, and C. Wong, *Classification Using Association Rules: Weaknesses and Enhancements.* Kluwer Academic Publishers, 2001, pp. 591–601.

[19] E. Loken and A. Gelman, "Measurement error and the replication crisis," *Science*, vol. 355, no. 6325, pp. 584–585, 2017.

[20] E. R. Omiecinski, "Alternative interest measures for mining associations in databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 1, pp. 57–69, Jan 2003.

[21] F. Padillo, J. M. Luna, and S. Ventura, "A grammar-guided genetic programing algorithm for associative classification in big data," *Cognitive Computation*, Jan 2019. [Online]. Available: https://doi.org/10.1007/s12559-018-9617-2

[22] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Proceedings of the 7th International Conference on Database Theory*, ser. ICDT '99. London, UK, UK: Springer-Verlag, 1999, pp. 398–416.

[23] R. Quinlan, *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[24] R. Quinlan and R. M. Cameron-Jones, *FOIL: a midterm report*, 01 1970, pp. 1–20.

[25] RFC. (2019) Common format for comma-separated values (csv) files. [Online]. Available: https://tools.ietf.org/html/rfc4180

[26] I. Triguero, S. González, J. M. Moyano, S. Garcîa, J. Alcalá-Fdez, J. Luengo, A. Fernández, M. J. del Jesús, L. Sánchez, and F. Herrera, "Keel 3.0: an open source software for multi-stage analysis in data mining," *International Journal of Computational Intelligence Systems*, vol. 10, no. 1, pp. 1238–1249, 2017.

[27] K. Wang, S. Zhou, and Y. He, "Growing decision trees on support-less association rules," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 265–269.

[28] A. Wiggins. (2019) The twelve-factor app. [Online]. Available: https://12factor.net/

[29] X. Yin and J. Han, "Cpar: Classification based on predictive association rules," in *3rd SIAM International Conference on Data Mining(SDM03)*, 2003, pp. 331–335.

[30] M. J. Zaki, "Scalable algorithms for association mining." *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.