Java V.S. Python in AI

Final Master Thesis – High Performance Computing

Master in Innovation and Research in Informatics
Facultat d'informàtica de Barcelona
Universitat Politècnica de Catalunya

Author:

Marco M. Aguado Acevedo

marcoaguado95@gmail.com

Supervisors:

Marisa Gil Gómez

marisa.gil@upc.edu

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

October 28, 2021

Acknowledgements

First and foremost, I would like to use this section to thank professor Marisa Gil for the opportunity she gave to me to work on this interesting project.

I would also like to thank professor Xavier for providing me access to the CTE-ARM system of the BSC and for providing advice regarding the analysis and the execution process.

Not only do I want to express my gratitude towards them for the unconditional support, but also for their patience and kindness.

I cannot end this note without thanking my whole family, especially my parents and aunt, who supported me throughout not only my academic career but also my life.

Abstract

Java's position as an appealing programming language for developing AI software is not clear in the current research area. It is well-stablished that other programming languages such as Python have grown in popularity in the AI research community as an easy to learn, easy to use and simple code tool over the more strict and complex languages such as Java. Also, with the increase in the number of processors with ARM architecture used for server purposes due to their better energy efficiency, the interest in their performance for specific research areas also increases. This thesis aims to provide an objective and up-to-date view of Java's current situation in this area by comparing it with Python both in terms of the coding process and their performance.

To test if Java could prove more value to researchers than Python, a set of representative AI algorithms were implemented in both languages using the nowadays most used libraries: Weka for Java and Scikit-learn, Pandas and Numpy for Python and executed in two different machines from the MareNostrum4 system: The Intel machine with x86 architecture processors and the CTE-ARM machine with ARM architecture processors using 4 different datasets with its own characteristics and sizes.

The results obtained show that Java performs worse than Python in almost all the tests by having higher execution times and worse memory management while the impact of using a machine with an ARM processor is minimal although in general it performs better than x86 processor, especially when comparing Java's performance in both machines.

These results suggest that Java does not provide enough value to because a more suitable programming language than Python for AI researchers as the obtained performance does not compensate the extra effort both in learning time and coding time Java requires to implement programs that solve the exact same problems.

Keywords: Java, Python, Artificial Intelligence, ARM, performance, code

Index

List of figures

1.Introduction

1.1 State of the art

Over the years, the AI technologies have taken a bigger part in our lives. From language recognition to the most advanced autonomous cars are possible thanks to the recent research of new and more efficient AI algorithms. Artificial Intelligence is a popular topic in 21st century, it is one of the advanced research fields of computer sciences. The work of AI began formally soon after World War II, and the word ”Artificial Intelligence” was put forward by Stanford professor John McCarthy in 1956 on Dartmouth workshop. The area of AI research goes further still, it aims not only exists in the theories but also to create entities, which includes a significant variety of subfield, scaling from universal (studying and feeling) to particular, such as playing chess, diagnosing diseases, writing poetry, driving vehicles and proof of mathematical theorems. With the technologies of AI evolving swiftly, its capabilities are being applied in all aspects of society.

Selecting the appropriate programing language is an important task for AI construction. There are diverse and general options such as python, C++ and LISP. Currently Python is one of the most popular programming languages for AI developers due to the simplicity of its code and the powerful AI libraries developed by the community such as Pandas, Numpy or Scikit-learn. Nevertheless, Java plays an important role in AI programming. Java is the most widely used programming language in the world, it is object-oriented and scalable, which is necessary for AI projects. Gigantic Java code bases are exploited by public and private organizations and sectors, and the Java Virtual Machine as a compute environment is heavily relied (Georges, et al., 2006). Java Virtual Machine allows developers to build a single application version that could run on all Java-enabled computing platforms. Maintainability, portability and transparency are the three main advantages of Java programming language.

However, there are two shortcomings of Weka compared with Python. First, Weka's pre-processing and result output are more difficult. Although it is convenient for beginners to process data with a little filter, it is easier to write programs like Python when processing large amounts of data (Caia, et al., 2005). Similarly, although the results can be run out by pressing "Start" in the classification, it is more troublesome for Weka to make the results lead to the format or the next application. Second, the Python package is booming. Although Weka also has a lot of packages, but a closer look will reveal that most of them are old and have not been updated. The Weka suite written in Java is also difficult to rewrite and compile. In contrast, the development of Python is flourishing. Most of late research could be repackaged into Python packages for people to download and use, and there are countless developers studying Python.

Previous studies have proven that in general purpose programs, java is the least memory efficient programming language (Prechel, s.f.) although the ones found are more than 5 years old. If research was made of the articles and/or research papers related to the role of Java in the AI world written in the last 4 years, one would find that there are almost none.

In the ARM environment, Java has a stretch relationship with the ARM architecture as Java was, until recent years, the main programming languages for Android developers and, therefore, for the ARM architecture used in this kind of devices with the Android operating system. Since the main objective behind Android development was to create a platform-independent application environment that can run on every device, Java was the perfect candidate to be the main programming language as it already has this quality.

However, there are not much information about the performance of Java and the JVM in the server market for the ARM processors. Thanks to the inclusion of ARM processors in MAC systems on November 10, 2020, some benchmarks have been made to test the performance of these systems although none of them test specifically java-based applications. The only benchmark testing the performance of a Java application on both ARM and x86 architecture found when this project was being made showed that ARM version consumed more memory but had in general a better performance (Voitylov, 2021).

Python have even less presence in the ARM architecture environment. Current research are made on FPGAs and the few results provided by the community suggest that Python have huge potential for this kind of environment (Schmidt, et al., 2017) with some stating that a 30x performance was obtained over the existing C code but the lack of implementation of certain libraries do not allow researchers to test new benchmarks.

1.2 Proposal

As more products and features are developed based on AI technology it is important to have a good base from which construct the system's structure. Therefore, having a good program language from which develop a solution is essential. As it was stated before, Python is the go-to program language nowadays for developing Ai software due to the easy coding and that anyone can quickly learn it but it is also important to also take in count other existing options as a base program language, in this case, Java.

Physics, mathematicians, and engineers are the main people developing new AI algorithms and solutions as they are they have a deep comprehension of the bases behind the AI algorithms and the statistical component of them. For them, Python is an appealing language program from which develop their solutions as it doesn't require a lot of time to learn its syntax and the final code produced is short in general. That is why many AI libraries such as NumPy of Sklearn are developed for Python. Java on the other hand has much less research behind in regard to AI libraries but there some of them like WEKA that are still used and simplifies the job of the programmer with its API. This is the reason that this project also aims to analyze the characteristics of both languages from the point of view of its code and specifically for the development of AI programs.

Performance is another key factor when comparing program languages. Both Java and Python have their own characteristics which in the end produce different results on their performance (Destefanis, et al., 2016). Both Java and Python are high level programming languages and so it is hard or even not possible to access the hardware directly to optimize code so they both depend on its virtual machine and the code implementation to determine its final performance. One of the most remarkable aspect is that most of the times, Python is interpreting bytecode and executing it locally while Java compiles to an "Intermediate Language" and the Java Virtual Machine reads the bytecode and just-in-time compiles it to machine code. This, together with the fact that Java is JIT-Compiled enables optimizations to be made at runtime. This JIT optimizer will see which parts of the application are being executed a large number of times and will then make optimizations to those bits of code, by replacing them with more efficient versions. It is also important to keep in mind that Java is strongly-typed language so the optimizer can make many more assumptions about the code.

With these premises it would be interesting to test the optimizations Java provides to have an objective perspective of its performance and in particular for AI algorithms.

It is also important to take in count and analyze other architectures such as ARM, which is increasing its use in the server, HPC and even in personal computers. ARM is the main processor architecture in the FUGAKU Supercomputer, the number 1 of the Top500 supercomputers in the world (Anon., s.f.) . The Barcelona Supercomputing Centre also includes an ARM machine and the Arm-BSC Centre of Excellence which aim to increase the collaboration between the BSC and ARM for research purposes. Therefore, this is an excellent opportunity to also take in count the architecture as a factor when comparing both languages.

1.3 Objectives

The aim of this project is to determine if Java still has a place in the AI world as a relevant programing language by comparing it to the nowadays most popular language in the AI software development, Python. Also, this thesis tries to provide and objective analysis of the ARM architecture with AI algorithms as the current state of the art is quite lacking in this regard.

The ultimate goal of the results is that they should provide a first approach of the actual performance of Java and Python using up to date technologies and be a possible starting point for future research regarding new techniques to increase Java performance for AI algorithms. Moreover, the output could highlight some possible unknown or unclear areas where Java could prove to be a more appealing tool for researchers.

The objectives tackled in the study are:

1. Evaluate the coding process for the implementation of most representative algorithms in Java and Python.
2. Evaluate the total length of code produced in the implementation of the algorithms.
3. Evaluate the performance of the implemented algorithms in Java and Python in a machine with x86 architecture.
4. Evaluate the performance of the implemented algorithms in Java and Python in a machine with ARM architecture.
5. Evaluate the impact of the architecture by analyzing the performance of the algorithms implemented in the same programing language but in a machine with different architecture.
6. Analyze the current state of Java on the AI world based on the previous evaluations.

2. Methodology

This section contains the research and analysis made over the different elements that will make part of the project as well as the metrics and characteristic that will be used to present results.

2.1 Study of the existing and most representative algorithms in the AI

AI environment contains many different areas of study and therefor, a study of the most representative algorithms was made. The algorithms selected can be classified into two categories: Regression and Classification.

2.1.1 Regression

Regression algorithm's role is to predict the output values of a certain problem based on input features from the data fed in the system. In order to do so, the algorithm builds a model based on the features of the training data and then using the model to predict the value for new data.

2.1.1.1 Linear regression

The linear regression algorithm is one of the most-used regression algorithms in Machine Learning. A significant variable or a group of them from the data set is chosen to predict an output variable. There are three types of regression algorithms: Simple Linear regression, Multilinear regression and the Non-linear Regression. In this work the Multilinear regression type will be used as it is more representative than the Non-linear regression (Yan & Su, s.f.) and has more computational complexity than the Linear regression.

Multilinear regression is used to estimate the relationship between two or more independent variables and one dependent variable. The algorithm analyses the data from a training set and creates a model with the following formula:

$$y = \beta_0 + X_1\beta_1 + \cdots + X_n\beta_n + \varepsilon$$

Where:

- y = the predicted value of the dependent variable
- β_0 = the y-intercept
- $X_1\beta_1$ = the regression coefficient of the first independent variable (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value)
- $X_n\beta_n$ = the regression coefficient of the last independent variable
- ε = model error (a.k.a. how much variation there is in our estimate of y)

2.1.2 Classification

Classification is defined as the process of recognition, understanding, and grouping of objects and ideas into pre-set categories. Classification algorithms used in machine learning utilize input training data for the purpose of predicting the probability that the data that follows will fall into one of the predetermined categories. From the existing classification algorithms, the following ones were analysed:

2.1.2.1 Logistic regression

Logistic regression is a mathematical modeling approach that can be used to describe the relationship of several independent variables X1, X2, . . . , Xk to a dependent variable and/or predict its value. To do so, it uses a logistic function, which describes the mathematical form on which the logistic model is based. The fact that the logistic function f(z) ranges between 0 and 1 is the primary reason the logistic model is so popular. The model is designed to describe a probability, which is always some number between 0 and 1.

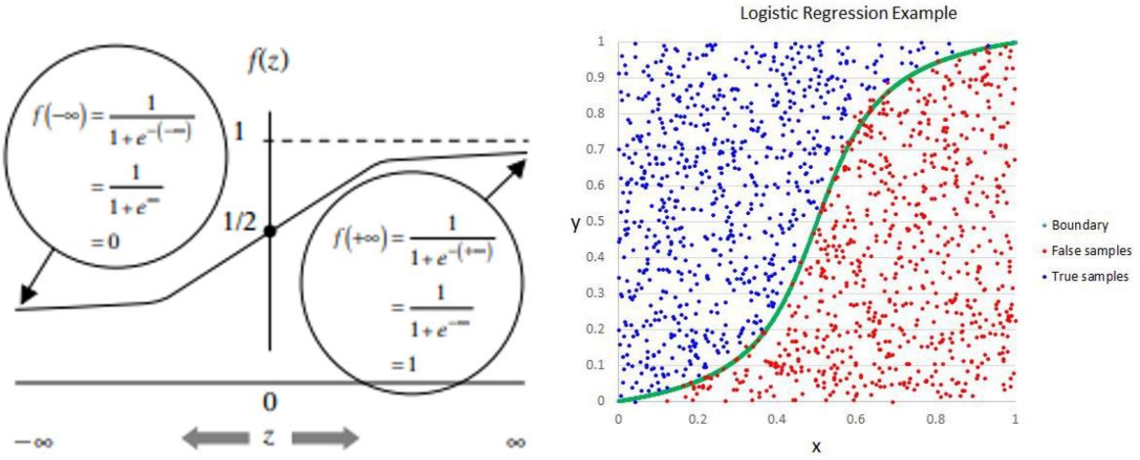


Figure 1: Logistic function and graphical interpretation

2.1.2.2 Support Vector Machine

Support Vector Machine (SVM) is one of the most extensively used supervised machine learning algorithms in the field of text classification. Given a set of points of 2 types in N dimensional place, SVM generates a (N — 1) dimensional hyperplane to separate those points into 2 groups. It computes the linear separation surface with a maximum margin for a given training set. When a linear separation surface does not exist, for example, in the presence of noisy data, SVMs algorithms with a slack variable are appropriate. This Classifier attempts to partition the data space with the use of linear or non-linear delineations between the different classes. The best hyperplane would be the one with the largest positive vectors and separating most of the data nodes. This is an extremely powerful classification machine that can be applied to a wide range of data normalization problems.

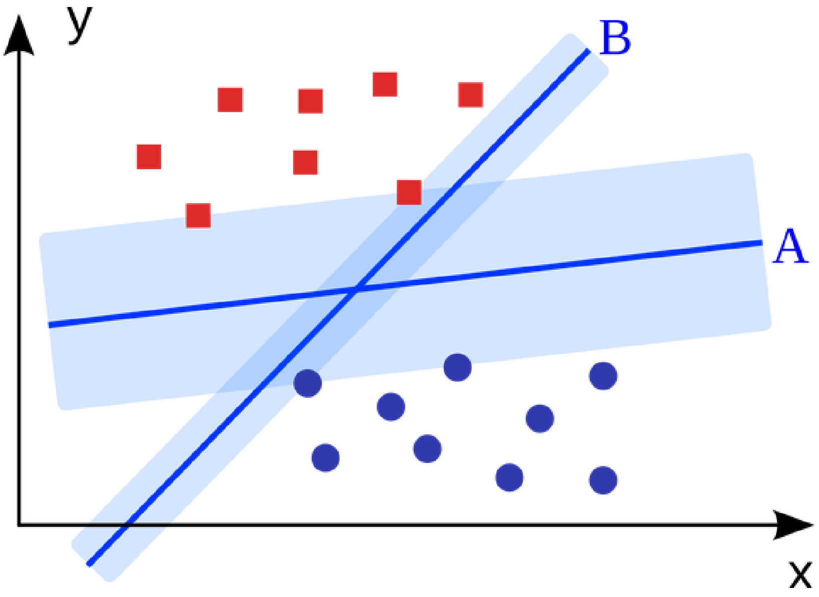


Figure 2: Hyperplane generated by a SVM

2.1.2.3 Decision Trees

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance-event outcomes, resource costs, and utility. In a Decision tree, there are three types of nodes, which are the Root Node, the Decision Node and the Leaf or Terminal Node. The Root node represents the entire population or sample and this further gets divided into two or more homogeneous sets, the Decision nodes are used to make any decision and have multiple branches, whereas Terminal nodes are the output of those decisions and do not contain any further branches.

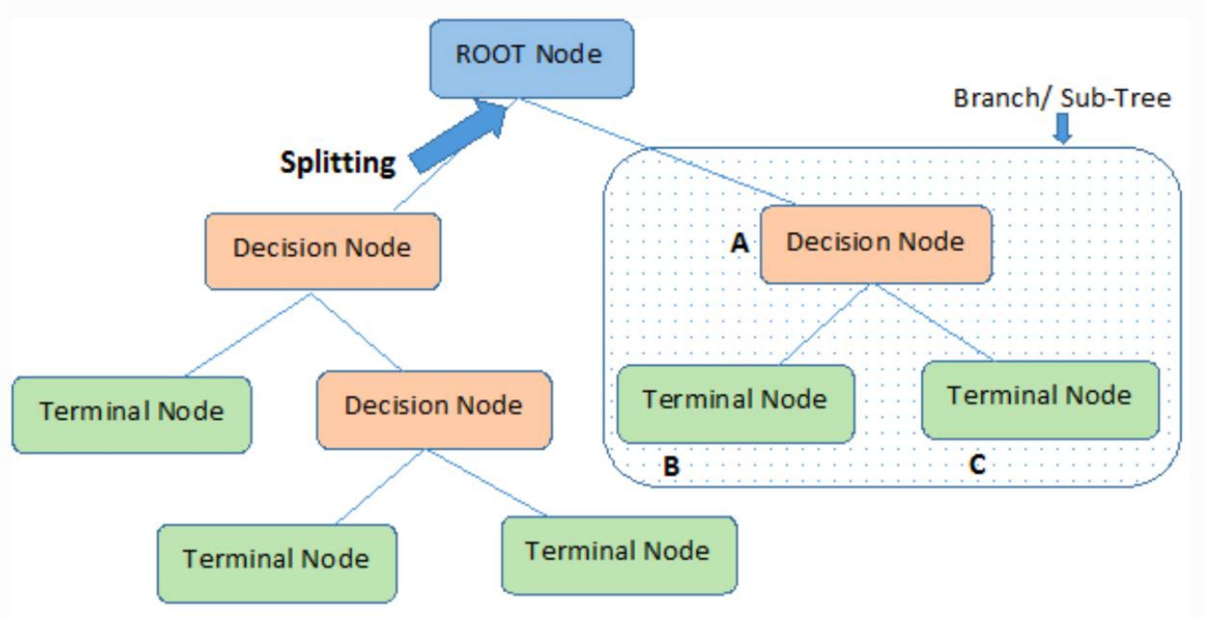


Figure 3: Decision Tree representing each of the nodes

In order to predict a class label for a record, the algorithm starts from the root of the tree. It compares the values of the root attribute with the record's attribute, and, on the basis of comparison, it follows the branch corresponding to that value and jump to the next node. More specifically, the complete process can be divided in the following steps:

1. Begin the tree with the root node, says S, which contains the complete dataset.
2. Find the best attribute in the dataset using Attribute Selection Measure (ASM).
3. Divide the S into subsets that contains possible values for the best attributes.
4. Generate the decision tree node, which contains the best attribute.
5. Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

2.1.2.4 k-Nearest Neighbor

k-Nearest Neighbor is a method that simply searches the closest observations to the one you are trying to predict and classifies the point of interest based on most of the data around it. It is a supervised algorithm so the training dataset is labeled, with the class or expected result given "a row" of data. It is also instance based so it does not explicitly learn a model (such as Logistic Regression or decision trees). Instead it memorizes the training instances that are used as a “knowledge base” for the prediction phase.

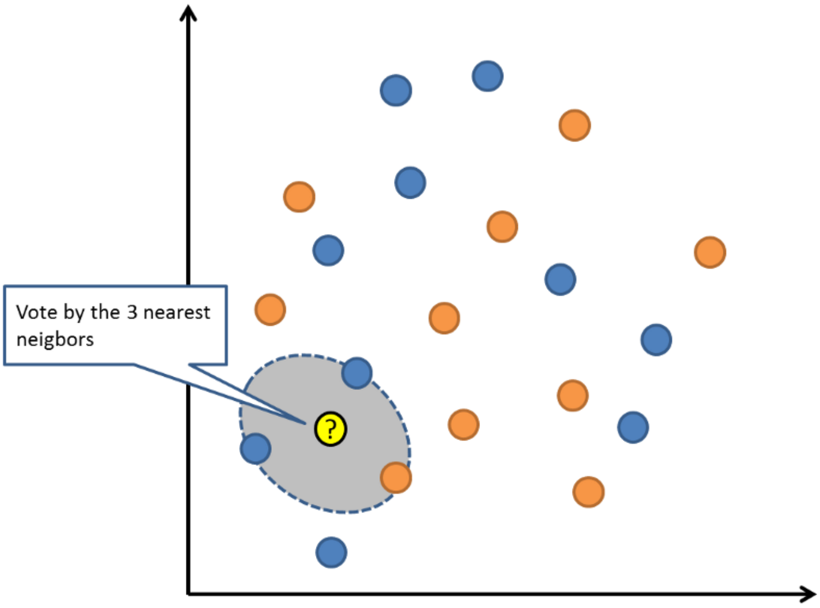


Figure 4: Graphical example of a KNN voting process

2.1.2.5 Naïve Bayes

A Naïve Bayes classifier is a probabilistic classifier based on Bayes theorem. It is a conditional probability model. The model is called naive as it operates on the assumption that all the input data values are unrelated to each other. While this cannot take place in the real world, this simple algorithm can be applied to a multitude of normalized data flows to predict results with a great degree of accuracy.

There are 3 main types of Naive Bayes algorithms:

- Gaussian: It is used in classification and it assumes that features follow a normal distribution.
- Multinomial: This one is used for discrete counts.
- Bernoulli or binomial: The binomial model is useful if the feature vectors are binary.

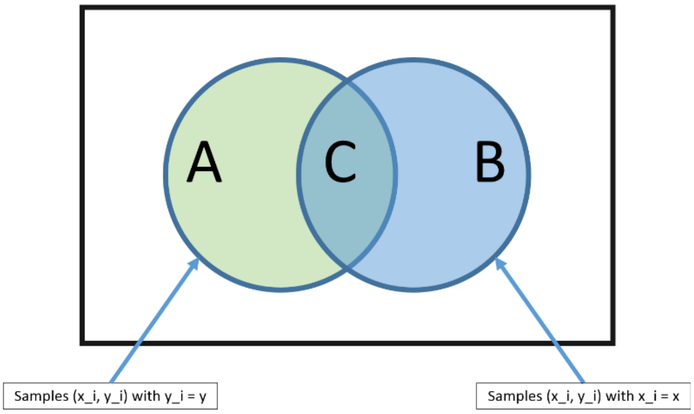

A
C
B
Samples (x_i, y_i) with y_i = y
Samples (x_i, y_i) with x_i = x

Figure 5: Naïve Bayes representation where the algorithm finds common features.

2.1.2.6 Random forest

The Random Forest algorithm consists of a large number of individual decision trees that operate as an ensemble. The algorithm establishes the outcome based on the predictions of the decision trees. Each individual tree in the random forest spits out a class prediction and the class with the most votes become the model prediction.

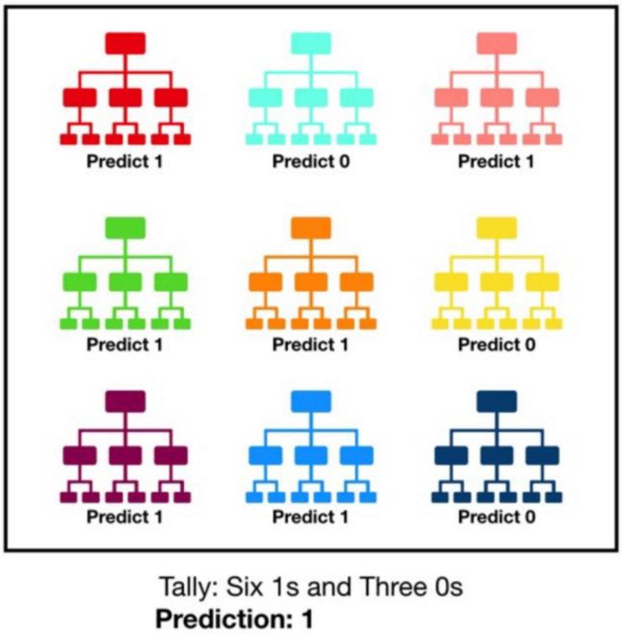


Figure 6: Random forest composed by 9 decision trees voting the class result.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the overfitting of datasets and increases precision.

2.2 Study of the available Hardware systems

BSC has provided two different accounts in order to fulfill this project.

An account for Marensotrum4 with an x86 architecture. Each computing module has the following characteristics:

- 2x Intel Xeon Platinum 8160 24C at 2.1 GHz
- 216 nodes with 12x32 GB DDR4-2667 DIMMS (8GB/core)
- 3240 nodes with 12x8 GB DDR4-2667 DIMMS (2GB/core)
- Interconnection networks:
- 100Gb Intel Omni-Path Full-Fat Tree
- 10Gb Ethernet
- Operating System: SUSE Linux Enterprise Server 12 SP2

An account for the MareNostrum4 CTE ARM machine

- A64FX CPU (1 Armv8.2-A + SVE chip) @ 2.20GHz (grouping the cores in 4 CMG - Core Memory group - with 12 cores/CMG and an additional assistant core per CMG for the Operating system, adding a total of 48 cores + 4 system-cores per node. The ARMv8.2-A cores have available the Scalable Vector Extension (SVE) SIMD instruction set up to 512-bit vector implementation.
- 32GB of main memory HBM2
- TofuD network
- Single Port Infiniband EDR
- The operating system is, Red Hat Enterprise Linux Server 8.1

2.3 Study of Java and Python programing languages

2.3.1 Java

Java is a high-level, object-oriented, general-purpose programming language widely used in personal computers, data centers, game consoles, supercomputers, mobile phones and the Internet applications. It was originally developed in 1991 by James Gosling, a Canadian computer scientist, at what was then Sun Microsystems, in the U.S. state of California. Today, after decades of effect, Java has been developed into a fully functional, multipurpose, and powerful language suitable for both individual and enterprise users.

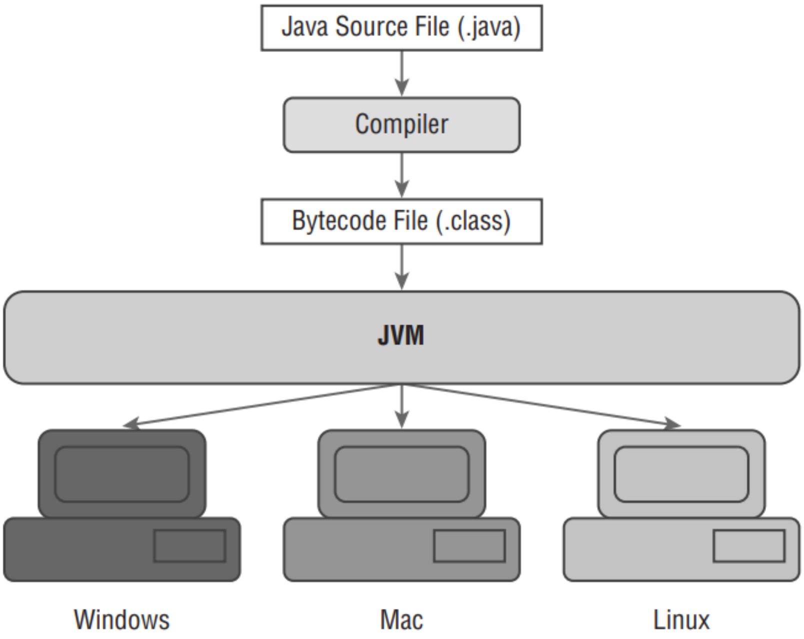


Figure 7: Java architecture structure

The main components of Java are: The Java Virtual Machine (JVM), the Java Runtime Environment (JRE) and the Java Development Kit (JDK).

The Java Virtual Machine (JVM) is the runtime engine of the Java Platform, which allows any program written in Java or other language compiled into Java bytecode to run on any computer that has a native JVM. It provides the functionality of important subsystems such as garbage collection, memory management and security. JVM is platform-independent and it can be customized and configured using a Virtual interface it provides which is not machine-dependent and is also independent of the operating system.

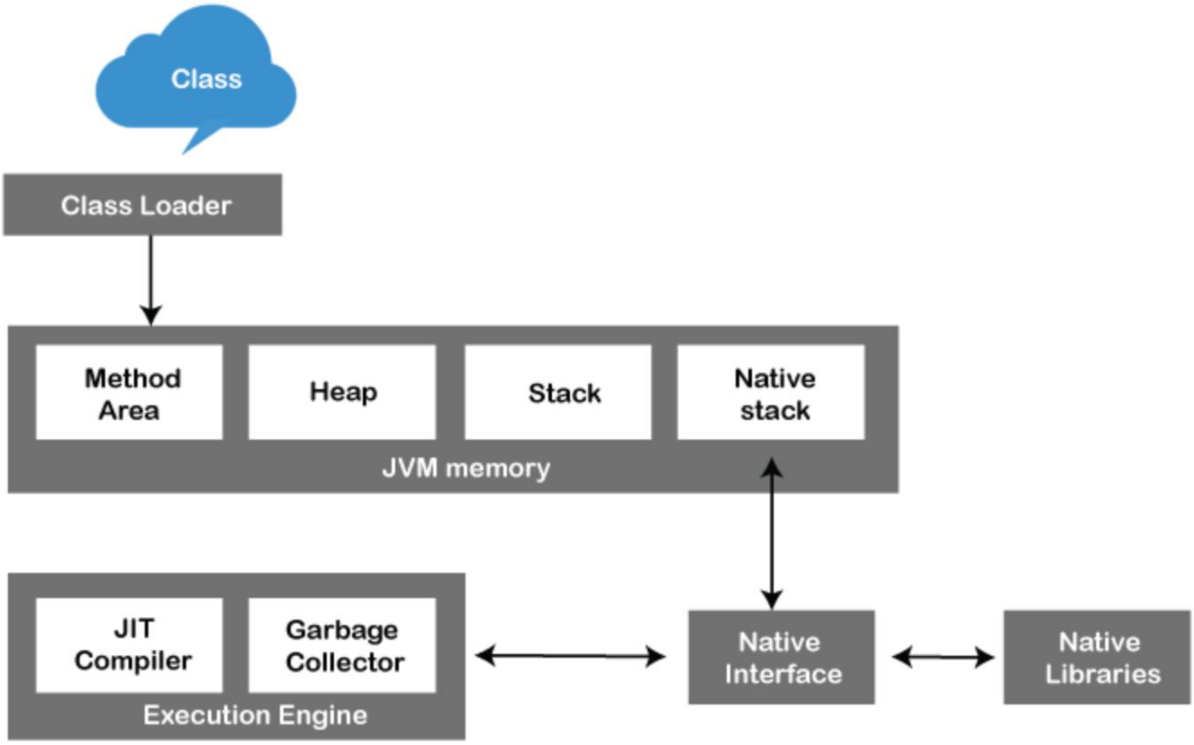


Figure 8: Java Virtual Machine components

The Java virtual machine

The main subsystems are:

- Class Loader: This is the subsystem of JVM used to load class files. Whenever a user run a java program, class loader loads it first.
- JVM memory subsystem: This is the subsystem of JVM that manages all the memory related aspects of the system. Its main parts are:

    - Class method area: It is one of the Data Area in JVM, in which Class data will be stored. Static Variables, Static Blocks, Static Methods, Instance Methods are stored in this area.
    - Heap: A heap is created when the JVM starts up. It may increase or decrease in size while the application runs.
    - Stack: It is also known as a thread stack and it is a data area in the JVM memory which is created for a single execution thread. The JVM stack of a thread is used by the thread to store various elements such as local variables, partial results, and data for calling method and returns.
    - Native stack: It subsumes all the native methods used in the application.

- Execution Engine: The execution engine is the Central Component of the JVM. It communicates with various memory areas of the JVM. Each thread of a running application is a distinct instance of the virtual machine's execution engine. Execution engine executes the byte code which is assigned to the run time data areas in JVM via class loader. Java Class files are executed by the execution engine. It also contains:

    - JIT compiler: The Just-In-Time (JIT) improves the performance of Java applications by compiling bytecodes to machine code at run time. The JIT compiler is enabled by default. When a method is compiled, the JVM calls the compiled code of that method directly. The JIT compiler compiles the bytecode of that method into machine code, compiling it just in time to run.
    - Garbage collector: Garbage Collector's task is to collect the unused data. It tracks each and every object available in the JVM heap space and removes unwanted or unused ones. Garbage collector works in two simple steps known as Mark, in which the garbage collector identifies which piece of memory is in use and which are not, and Sweep, in which the garbage collector removes objects identified during the "mark" phase

The Java Runtime Environment is the runtime environment that is required to execute Java programs and applications. JRE consists of the JVM, binaries and other classes that contains and orchestrates the set of tools and minimum requirements for executing a Java application. JRE is a subset of JDK and doesn't contain any development tools such as Java compiler or debugger. The JRE includes the following components:

- Code libraries, property settings, and resource files.
- DLL files: Used by Java hotspot client virtual machine and server virtual machine.
- Java extension files
- Files required for security management such as java.policy, java.security

- Applet support classes
- True Type font files

The Java Development Kit is the core component of a Java environment. It is a software development environment used to program Java applications and applets. As it was stated before, it contains JRE and several development tools, an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) accompanied with another tool. JDK contains the following components:

- jConsole: This is a Java monitoring and management Console.
- jar: This is the archiver. This tool is used to package related class libraries into a single Jar file as well as to manage Jar files.
- jarSigner: This tool is used for jar signing and verifying.
- javap: This is a tool for class file disassembler.
- javaws: Java web start launcher for JNLP applications.
- jhat: Java heap analysis tool.
- jrunscript: Java command-line script shell.
- jstack: Utility used to print stack traces for Java threads.
- Javadoc: This automatically generates documentation from the source code comments.
- appletviewer: Used for applet execution and debugging without a web browser.
- apt: Annotation processing tool.
- extCheck: Utility used to check jar file conflicts.
- keytool: Using this utility you can manipulate Keystore.
- policytool: This is a policy creation and management tool.
- xjc: This is a part of the XML binding (JAXB) API that accepts XML schema and generates Java classes.

2.3.2 Python

Python is an interpreted, multipurpose, object-oriented, high-level programming language with dynamic semantics. It was created by Guido van Rossum, and first released on February 20, 1991. Python uses code modules that are interchangeable instead of a single long list of instructions that was standard for functional programming languages.

The main components of the Python programming language are the interpreter and a support library.

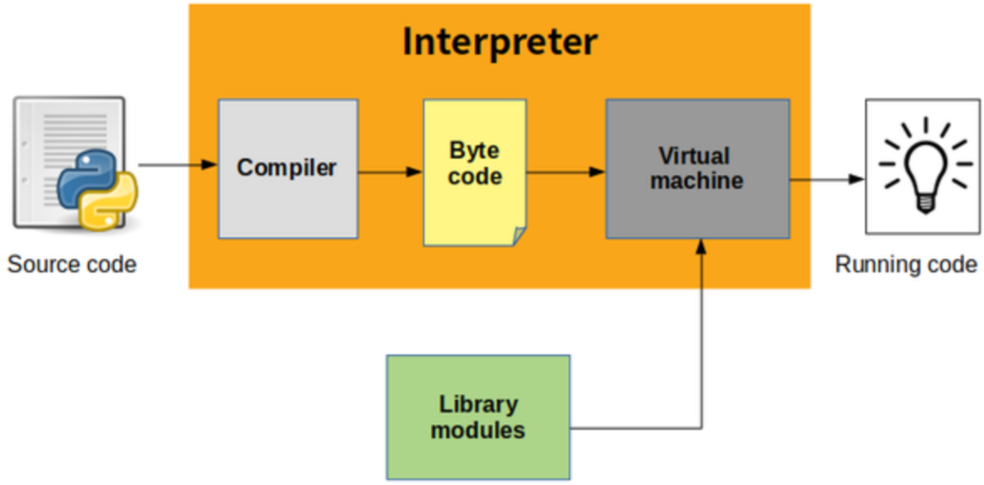


Figure 9: Python execution process

When a python code is executed, it is firstly compiled into python byte code, which creates file with extension .pyc . The byte code compilation happened internally, and almost completely hidden from developer. Compilation is simply a translation step, and byte code is a lower-level, and platform-independent , representation of your source code. Each of the source statements is translated into a group of byte code instructions. This byte code translation is performed to speed execution so this byte code can be run much quicker than the original source code statements.

The .pyc file created in compilation step is then executed by the Python virtual machine which is the runtime engine of Python and it is always present as part of the Python system.The Virtual Machine iterates through each of the byte code instructions to carry out their operations.

2.4 Study of the characteristics and metrics to compare

In order to do a solid comparison between Java and Python there are some metrics that must be stablished, measured and analyzed.

2.4.1 Code

This represents how easy and time coming is to code AI programs in both Java and Python as well as the extension of the final code.

One of the factors that needs to be compared when analyzing two programming languages is the syntax and the code. There are some previous work that compare the main characteristics of both languages (Khoirom, et al., 2020) although they only show a general representation of its syntax. Here the idea is to go one step further and take a look at the specific implementation of Ai algorithms in both languages to see it Java proves to be an appealing tool for researchers.

This analysis looks at both Java and Python's code syntax and rules as well as the libraries used and compares the difficulties to code the algorithms in terms of the previous knowledge required, usability of the code, organization and code length.

2.4.2 Performance

2.4.2.1 Execution time

It is defined as the total time it takes for a program to finish its task. There are three types of execution time:

- Worst-case execution time (WCET): the longest execution time for any possible combination of inputs
- Best-case execution time (BCET): the shortest execution time for any possible combination of inputs
- Average execution time: case execution time for typical inputs

For the purpose of this project, the Execution time used will be the Average execution time since it is the most suited to find if the performance obtained of a certain program is due to poor algorithms, bad coding, poor choice of instructions, or other causes (Wolf, 2014).

2.4.2.2 CPU efficiency

CPU efficiency is another key aspect of a program's performance. It is defined as how efficient an application is when utilizes its requested CPU time. Only the CPU time is taken in count. This project does not take in count the power efficiency of the CPU.

It is calculated as (IBM, s.f.):

$$CPU\ efficiency = \frac{CPU\ time}{Run\ time * Number\ of\ CPUs}$$

2.4.2.3 Memory usage

A correct and efficient usage of memory is also an important characteristic to analyze. Here, the total amount of memory used by the program is measured and compared.

2.4.2.4 Time/Memory tradeoff

It is common that the programs that use more memory have better results in terms of execution time (Bonakdarpour, et al., 2011). Although this rule doesn't always occur, it is also interesting to test this relation for this specific environment and program languages.

2.5 Study and research of existing AI libraries

2.5.1 Python

2.5.1.1 NumPy

NumPy stands for Numerical Python. It is a library consisting of multidimensional array objects and a collection of routines to perform mathematical and logical operations in those arrays.

Figure 10:NumPy logo

A NumPy array is a multidimensional and uniform collection of elements. Unlike matrices, NumPy arrays can have any dimensionality. Furthermore, they may contain other kinds of elements or even combinations of elements, such as Booleans, dates or other (van der Walt, 2011).

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open-source project.

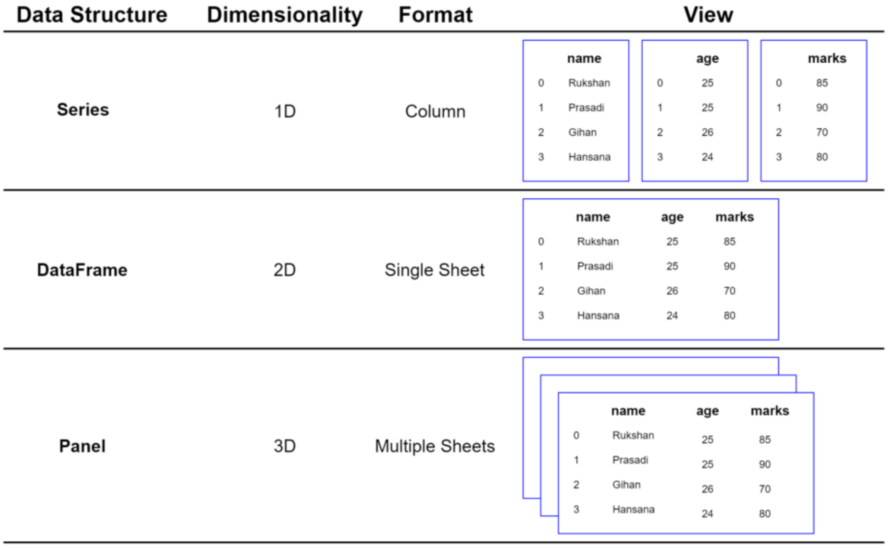
2.5.1.2 Pandas

Pandas is a Python library for data analysis. It was created by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool. Pandas has grown into one of the most popular Python libraries. It has an extremely active community of contributors.

Figure 11: Pandas logo

Pandas is built on top of two core Python libraries: Matplotlib, for data visualization and NumPy, for mathematical operations. Pandas acts as a wrapper over these libraries, allowing the programmer to access many of Matplotlib's and NumPy's methods and therefor, simplifying the code.

Before pandas, most analysts used Python for data munging and preparation, and then switched to a more domain specific language like R for the rest of their workflow. In order to eliminate the need to switch tools, Pandas introduced two new types of objects for storing data that make analytical tasks easier: Series, which have a list-like structure, and DataFrames, which have a tabular structure. It also possesses a Panel object to store data in a 3D structure.



Data Structure	Dimensionality	Format	View
Series	1D	Column	name 0 Rukshan 1 Prasadi 2 Gihan 3 Hansana age 0 25 1 25 2 26 3 24 marks 0 85 1 90 2 70 3 80
DataFrame	2D	Single Sheet	name age marks 0 Rukshan 25 85 1 Prasadi 25 90 2 Gihan 26 70 3 Hansana 24 80
Panel	3D	Multiple Sheets	name age marks 0 Rukshan 25 85 1 Prasadi 25 90 2 Gihan 26 70 3 Hansana 24 80

Figure 12: Pandas data structures

2.5.1.3 Scikit-Learn

Scikit-learn or Sklearn is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Figure 13: Scikit-Learn logo

This library was originally called scikits.learn and was initially developed in 2007 by David Cournapeau as a Google summer code project. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from the French Institute for Research in Computer Science and Automation, took this project at another level and made the first public release on the 1st Feb. 2010.

It is one of the most used ML libraries in Python as is both well-documented and easy to learn and use. As a high-level library, it lets the programmer to define a predictive data model in just a few lines of code, and then use that model to fit the data.

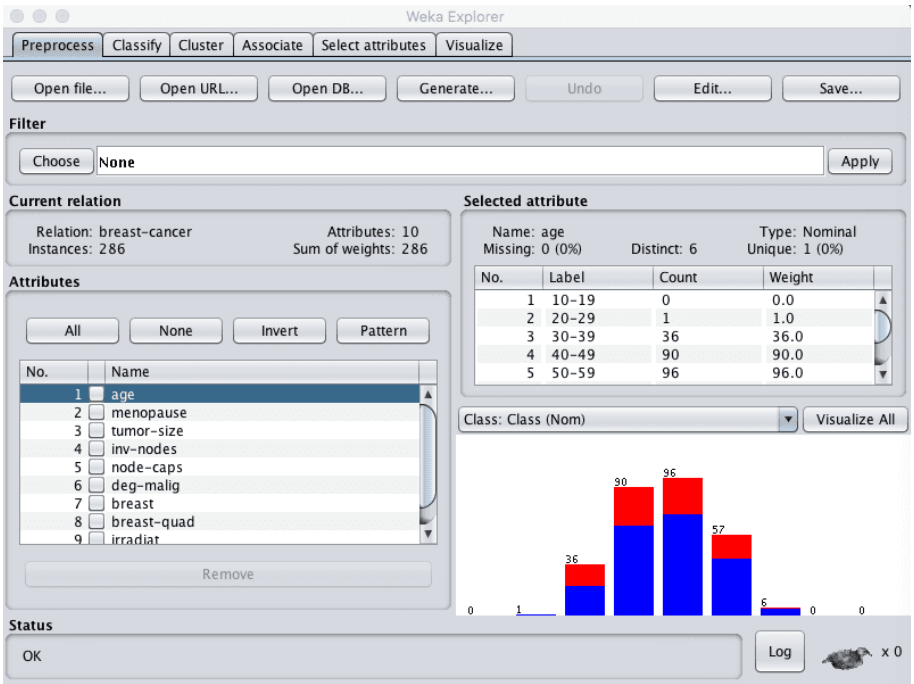
2.5.2 Java

2.5.2.1 Weka

Weka stands for the Waikato Environment for Knowledge Analysis. It is a workbench software written in Java, developed at University of Waikato in New Zealand that contains a collection of machine learning algorithms for data mining tasks.

Although it was created in 1993 and it is not updated as frequently as other AI tools, WEKA is still the most used programing and analysis tool used by Java developers and researchers (Joshi, 2018) to the point that even new developed tools such as Deeplearning4j uses the implemented algorithm classes in the Weka library to operate.

It provides a GUI which researchers can use to analyze and apply the provided algorithms and processing data tools without the need to directly code the solution and also an API to develop all the process or implement new versions of the existing algorithms and solutions. The user selects a learning method using the interactive interface menu. Most learning programs have adjustable parameters. The user can modify the parameters through the attribute list or object editor, and then evaluate the performance of the learning scheme through the same evaluation module.



Filter

Choose None

Current relation

Relation: breast-cancer
Instances: 286
Attributes: 10
Sum of weights: 286

Selected attribute

Name: age
Missing: 0 (0%)
Distinct: 6
Type: Nominal
Unique: 1 (0%)

No.	Label	Count	Weight
1	10-19	0	0.0
2	20-29	1	1.0
3	30-39	36	36.0
4	40-49	90	90.0
5	50-59	96	96.0

Attributes

No.	Name
1	□ age
2	□ menopause
3	□ tumor-size
4	□ inv-nodes
5	□ node-caps
6	□ deg-malig
7	□ breast
8	□ breast-quad
9	□ irradiat

Class: Class (Nom)

Status

OK

Figure 14: Weka GUI

Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

Thanks to the implementation of the weka library and environment as a JAR file, it is easy and simple for the developer to dynamically import the library into the code and link all the software together. It also provides a package manager system which allows to easily install or uninstall some extra libraries for specific algorithm implementations without the need to manage different JAR files. This is quite useful to create a solid and at the same time clean software environment to develop data analysis applications.

2.5.2.2 Other existing AI libraries for Java

There are other AI libraries coded in Java and available for the developers:

- Java Machine Learning (Java-ML)
- Deep Learning for Java (DL4J)
- CognitiveJ

These were not selected as they require their own framework and do not have such a strong presence as Weka in the Java AI research area.

2.6 Study and research of possible monitoring tools

Monitoring and analyzing the performance of a program depends in great measure from the tool selected. Both Python and Java languages provide tools to monitor and analyze the performance of programs written in said languages.

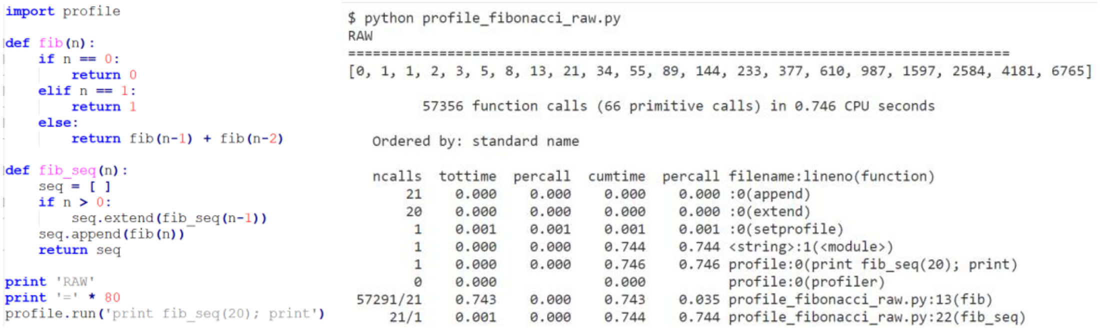
2.6.1 Python

Python has its own profiler which allows the programmer to obtain performance data of its programs in high detail (Gocht, et al., s.f.). The Python standard library provides two different implementations of the same profiling interface:

- Profile: It is a pure Python module but adds significant overhead to profiled programs.

- cProfile: it's a C extension based on lsprof with reasonable overhead that makes it suitable for profiling long-running programs.

Both profiles allow to create traces of the executing code and give parameters of the performance of each aspect of the code such as specific functions or sections of the code. It provides a small amount of functions to profile and analyze code which are also very easy to use as shown in the following example with a small Fibonacci program:



```python
import profile

def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def fib_seq(n):
    seq = [ ]
    if n > 0:
        seq.extend(fib_seq(n-1))
    seq.append(fib(n))
    return seq

print 'RAW'
print '=' * 80
profile.run('print fib_seq(20); print')
```

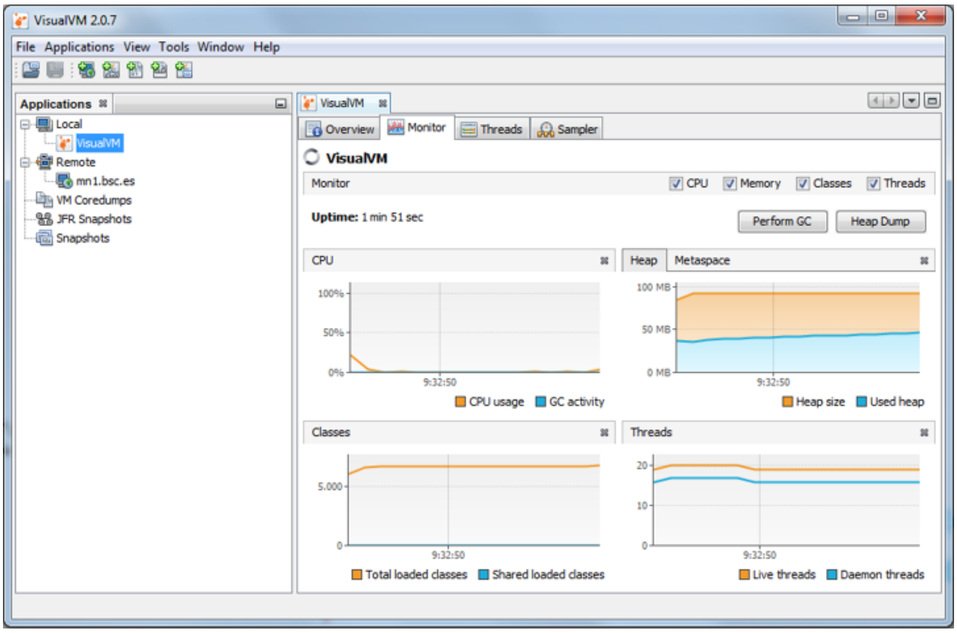
```
$ python profile_fibonacci_raw.py
RAW
================================================================================
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765]

         57356 function calls (66 primitive calls) in 0.746 CPU seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       21    0.000    0.000    0.000    0.000 :0(append)
       20    0.000    0.000    0.000    0.000 :0(extend)
        1    0.001    0.001    0.001    0.001 :0(setprofile)
        1    0.000    0.000    0.744    0.744 <string>:1(<module>)
        1    0.000    0.000    0.746    0.746 profile:0(print fib_seq(20); print)
        0    0.000             0.000          profile:0(profiler)
 57291/21    0.743    0.000    0.743    0.035 profile_fibonacci_raw.py:13(fib)
     21/1    0.001    0.000    0.744    0.744 profile_fibonacci_raw.py:22(fib_seq)
```

Figure 15: Python profiler example

As it is shown in the image above, by only adding the profile.run at the end of the code it is possible to obtain some detailed information about the performance obtained in the program execution.

2.6.2 Java

Oracle has developed a monitoring tool called JMX for the Java Virtual Machine that allows the programmer to monitor the state of the JVM, which resources are being used by it and all of its parts including the garbage collection. It also provides a UI to graphically monitor the state of the JVM and the resources used by it both in real time and for specific runs of a java program in the system.



Uptime: 1 min 51 sec

Figure 16: JMX interface

This tool also allows to monitor a specific JVM in a remote system and take snapshots of specific time intervals.

It is the most accurate tool available to measure a java program's performance (Fries, 2012) as it is the most complete and provides information that other systems may ignore such as the resource usage of the garbage collector system. Unfortunately, it was not possible to use this tool in the current MareNostrum4 and it was discarded.

2.6.3 System provided tools

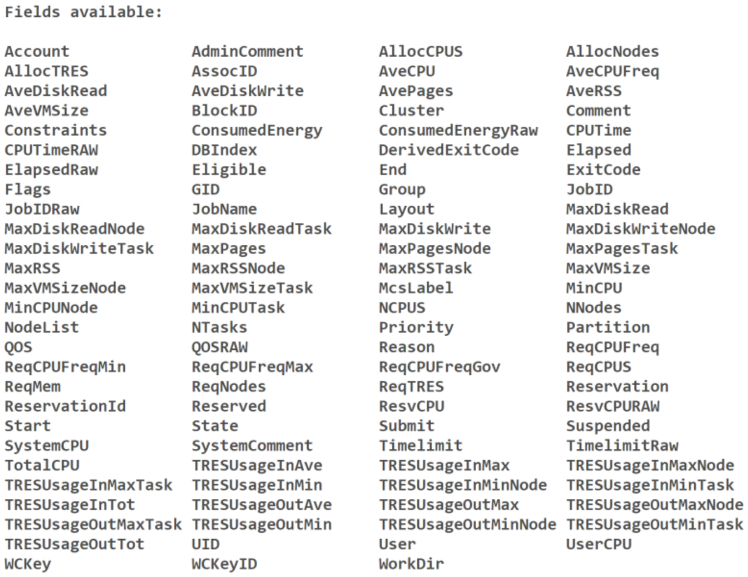
MareNostrum4 x86 SLURM

SLURM is the job scheduler system implemented in MareNostrum4 to launch and monitor the performance of the programs.

Figure 17: SLURM logo

SLURM provides some tools to measure the performance and the resources used for every job launched by a user. From the two existing commands to obtain the performance of the finished jobs, only sacct is available so this is the one used.

The sacct command displays job accounting data stored in the job accounting log file or Slurm database in a variety of forms for the analysis. The sacct command displays information on jobs, job steps, status, and exitcodes by default but there are several other statistics that can be obtained and are the following:



```
Fields available:

Account             AdminComment        AllocCPUS           AllocNodes
AllocTRES           AssocID             AveCPU              AveCPUFreq
AveDiskRead         AveDiskWrite        AvePages            AveRSS
AveVMSize           BlockID             Cluster             Comment
Constraints         ConsumedEnergy      ConsumedEnergyRaw   CPUTime
CPUTimeRAW          DBIndex             DerivedExitCode     Elapsed
ElapsedRaw          Eligible            End                 ExitCode
Flags               GID                 Group               JobID
JobIDRaw            JobName             Layout              MaxDiskRead
MaxDiskReadNode     MaxDiskReadTask     MaxDiskWrite        MaxDiskWriteNode
MaxDiskWriteTask    MaxPages            MaxPagesNode        MaxPagesTask
MaxRSS              MaxRSSNode          MaxRSSTask          MaxVMSize
MaxVMSizeNode       MaxVMSizeTask       McsLabel            MinCPU
MinCPUNode          MinCPUTask          NCPUS               NNodes
NodeList            NTasks              Priority            Partition
QOS                 QOSRAW              Reason              ReqCPUFreq
ReqCPUFreqMin       ReqCPUFreqMax       ReqCPUFreqGov       ReqCPUS
ReqMem              ReqNodes            ReqTRES             Reservation
ReservationId       Reserved            ResvCPU             ResvCPURAW
Start               State               Submit              Suspended
SystemCPU           SystemComment       Timelimit           TimelimitRaw
TotalCPU            TRESUsageInAve      TRESUsageInMax      TRESUsageInMaxNode
TRESUsageInMaxTask  TRESUsageInMin      TRESUsageInMinNode  TRESUsageInMinTask
TRESUsageInTot      TRESUsageOutAve     TRESUsageOutMax     TRESUsageOutMaxNode
TRESUsageOutMaxTask TRESUsageOutMin     TRESUsageOutMinNode TRESUsageOutMinTask
TRESUsageOutTot     UID                 User                UserCPU
WCKey               WCKeyID             WorkDir
```

Figure 18: Sacct statistics

2.6.4 MareNostrum4 CTE-ARM System

The CTE ARM system uses a different system to schedule and manage jobs and therefor the previous SLURM system can’t be used to obtain the data required.

2.7 Implementation of the code and preparation of data

2.7.1 Implementation of the algorithms

As it was stated before, the implementation of the final code relies on the selected libraries. This is because they are the most used by both Java and Python communities and therefore the most representative which will make the results obtained a more solid representation of the current performance of Java and Python in AI.

2.7.1.1 Structure of the code

In both regression and classification implementations, the program structure follows the same structure which consists of three parts:

- Data loading: This first part collects the information from the datasets and organize it in the different attributes and values corresponding to the different elements of the provided file.
- Model creation: In this section a specific model is created for the desired algorithm that is going to be tested. In this process the training data is provided to the model in order to fit and train it so it can try to predict or classify the testing data depending on if the algorithm is a regression or a classification one.
- Execution and testing: The final part of the program takes the previously created model and the testing data, generates its own regression or classification prediction and compares it with the real testing data.

2.7.1.2 ARM Python special case

Currently there are no Pandas and/or NumPy available libraries implemented for the ARM architecture and therefor the Python analysis in the ARM architecture cannot be made following the same structure as the previous ones. However, an implementation of the Linear Regression and Naïve Bayes algorithms were made from scratch using the basic Python libraries in order to provide a view of its performance on this architecture. It must be noted though, that it won’t provide the same performance as the x86 ones since those use the Pandas and NumPy libraries which have been optimized over many years by the community.

2.7.2 Dataset selection and preprocessing

Building an AI system normally involves sourcing large amounts of data and creating data sets for training, testing and evaluation. Data availability will have a major impact on how the system is assembled and what AI techniques will be used. The quantity and quality of data available will have an impact on the quality of the final product. This means that data availability and accessibility are the key elements to develop products that use AI technologies.

There exist a large variety of repositories with datasets available to the public. The task of selecting a suitable dataset for an AI system requires a considerable amount of time and research.

Data preprocessing is a key step when dealing with an AI application. It allows the researcher to analyze the data and select which attributes are more related with a specific problem. It consists of cleaning the data and getting it ready for training. Normally this is a time-consuming process and it involves removing data that is likely to skew results while retaining enough noise in the data to avoid overfitting. It is important to get to know the data well and how to best organise it to enable learning. Not understanding how the data was put together and how it works with the processing models, will reduce the effectiveness of the AI system. This whole process not only optimizes the performance of the AI application due to the reduced number of unrelated or redundant data, but it also provides a more accurate final result since the unwanted data won't interfere with the rest of the elements.

Each AI algorithm has its peculiarities and therefor are more suited for specific types and sizes of datasets. Due to the nature of the final code, the size of the data should be fairly big as this will give a clearer view of the possible differences between each implementation in terms of its performance. This means that each algorithm requires its own analysis to select and preprocess its data. Each dataset is divided into training and testing at a rate of 80/20 meaning that 80% of the dataset is used for training and the 20% for testing.

After several research, data-processing and testing, the following datasets were the ones used:

2.7.2.1 CalCOFI dataset

The CalCOFI dataset contains the longest (1949-present) and most time series of oceanographic and larval fish data in the world. It includes abundance data on the larvae of over 250 species of fish; larval length frequency data and egg abundance data on key commercial species; and oceanographic and plankton data. The whole dataset is represented by the bottle.csv which was obtained from Kaggle, a webpage with a large and diverse database of datasets.

This dataset possesses the suitable characteristics for a multi-linear regression model as it provides 74 columns of data and almost all of them are numeric values. As it was stated before, Linear regression is the only regression algorithm tested in this thesis and therefore, it requires a huge dataset consisting entirely of numeric variables since quantitative variables makes no sense for a regression problem.

The preprocessing for this specific dataset consisted of eliminating the columns containing IDs as they don't provide value to the prediction and the columns with more than 30% on null rows as well as filling the rest of null values by calculating the mean value of the column and filling the empty spaces.

The selected variable to predict is Salnity, which represents the salinity of the sea and can be estimated from the rest of the remaining dataset after it has been preprocessed.

2.7.2.2 Heterogeneity Activity Recognition dataset

The Heterogeneity Activity Recognition dataset is divided in two parts which contains the readings of two motion sensors, an accelerometer and gyroscope, from a set of 8 smartphones (2 Samsung Galaxy S3 mini, 2 Samsung Galaxy S3, 2 LG Nexus 4, 2 Samsung Galaxy S). Readings were recorded while users executed activities scripted in no specific order carrying the smartphones. These activities are 'Biking', 'Sitting', 'Standing', 'Walking', 'Stair Up' and 'Stair down'. From the previous two datasets, the gyroscope dataset was the one used here.

The dataset contains 43930257 rows and 10 columns which are the following:

- Index: The index of the specific reading
- Arrival_Time: The time the reading arrived to the gyroscope counting from the start of the experiment measured in seconds.
- Creation_Time: The time the current experiment was created measured in Day/Month/year.
- X: The ‘x’ coordinate of the device when the reading was taken.
- Y: The ‘y’ coordinate of the device when the reading was taken.
- Z: The ‘z’ coordinate of the device when the reading was taken.
- User: ID of the user from whom the reading was taken
- Model Device: The mobile device used to take the reading.
- gt: This represents the activity the user is doing when the reading was taken.

This dataset was suitable for a classification algorithm using the ‘gt’ column as the objective attribute to classify. The objective then would be to try to classify the activity a user is doing based on the position of a user for each reading taken.

The preprocessing of this dataset required to eliminate the columns that do not provide useful information to predict the desired attribute. In this case the 'Index', 'Arrival_Time', 'Creation_Time', 'User' and 'Model 'Device' columns were eliminated.

The resulted dataset consisted of a large amount of rows and a reduced number of columns which was suitable dataset structure for the Decision Tree algorithm as it reduces the amount of decisions the algorithm needs to make for each instance while the number them allowed to provide enough computational weight to enhance the possible performance differences between Java and Python.

2.7.2.3 Physical Unclonable Functions dataset

This dataset was generated from k-XOR Arbiter PUFs simulation using 5-XOR arbiters of 128bit stages PUF. It consists of 1 million rows and 129 attributes. The first 128 attributes represent the 128 bits of the XOR arbiter and the last attribute is the class label ‘result’ with values 1 or -1 representing the value obtained after applying the 128-XOR arbiters.

In this case, the dataset did not require a preprocessing step since every attribute is needed to estimate the class attribute.

The structure of this dataset contains multiple attributes and instances, and the class attribute has only two possible values. Those two characteristics make this dataset suitable for the Logistic regression algorithm as well as the Naïve Bayes algorithm. On one part the Logistic Regression algorithm is not affected by huge number of instances, and it is designed for class values with two possible outcomes and on the other part the Naïve Bayes algorithm is optimal when the dataset is considerably large.

2.7.2.4 Product Analytics dataset

This dataset contains the information of an online store of sporting goods. More specifically it contains a list of users to whom a banner was sent showing a product of the store and whether if they clicked on it and/or they finally bought the product. The dataset was also obtained from Kaggle. It contains 8471220 instances and 8 attributes which are the following:

- Order_id: A unique purchase number
- User_id: A unique customer identification

- Page_id: A unique page number for the event bundle
- Product: The name of the product offered
- Site_version: The device from which the banner was showed: mobile or laptop
- Time: The date the banner was showed to the user
- Title: This identifies the event type, meaning that the banner was clicked or only showed (not clicked)
- Target: This attribute shows if the user finally purchased the product (1) or not (0)

Half of the attributes of this dataset were deleted during its preprocessing as the IDs of a dataset almost never provide useful information in a classification process and, in this case, neither the moment the banner was showed. Therefore, the final dataset consisted of 4 attributes including the class attribute.

The characteristics of this dataset make it suitable for the Support Vector Machine algorithm since the dataset only has two possible values for the class attribute, meaning that the hyperplane created can classify the elements easier and then obtaining a higher accuracy. This dataset is also suitable for the k-Nearest Neighbor algorithm. This is because the KNN is quite sensible to the dimensionality of the problem, making this small dimension dataset appropriate. Also, the data is normalized which is recommended when using the the KNN as it helps identify the nearest neighbour characteristics for each instance.

2.7.2.5 CSV and ARFF extensions

Another aspect that is important to take in count is the file format used in both Java(Weka) and Python (NumPy/Pandas).

Comma-separated value or CSV files are the most popular formats for storing tabular data generated by IoT systems accustomed to store tabular data, like a spreadsheet or database. In a .csv file, the values of the records are stored in plain-text rows, with each row containing the values of the fields separated by a separator. The separator is a comma by default but can be configured to be any other character. This file format is really useful to represent the different attributes (in columns) and instances (rows) of a dataset.

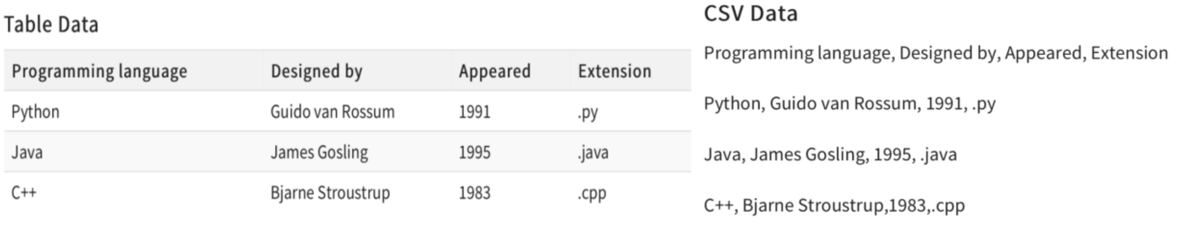


Table Data

Programming language	Designed by	Appeared	Extension
Python	Guido van Rossum	1991	.py
Java	James Gosling	1995	.java
C++	Bjarne Stroustrup	1983	.cpp

CSV Data

```
Programming language, Designed by, Appeared, Extension

Python, Guido van Rossum, 1991, .py

Java, James Gosling, 1995, .java

C++, Bjarne Stroustrup,1983,.cpp
```

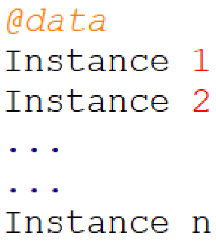
Figure 19: CSV representation of a dataset

ARFF files (Attribute-Relation File Format) are the most common format for data used in Weka. Each ARFF file must have a header describing what each data instance should be like and a Data section that contains each instance and the actual information. The attributes that can be used are the following:

- Numeric: Real or integer numbers.
- Nominal: Nominal attributes must provide a set of possible values.

- String: Allows for arbitrary string values. Usually processed later using the StringToWordVector filter.
- Date: Allows for dates to be specified. As with Java's SimpleDateFormat, this date can also be formatted; it will default to ISO-8601 format.

The ARFF Data section of the file contains the data declaration line and the actual instance lines. It is denoted in the file by the @data declaration. The format is:



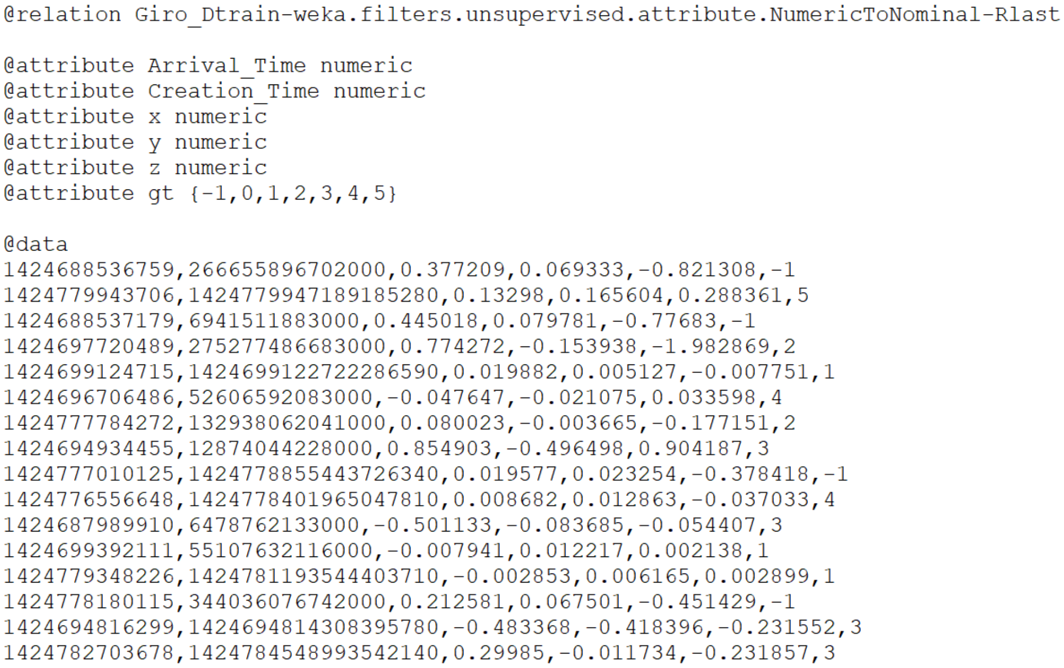
```
@data
Instance 1
Instance 2
...
...
Instance n
```

Figure 20:

Each instance is represented on a single line, with carriage returns denoting the end of the instance. A percent sign (%) introduces a comment, which continues to the end of the line.

Attribute values for each instance can be delimited by commas or tabs. A comma/tab may be followed by zero or more spaces. It is important that the attribute values must appear in the order in which they were declared in the header section. A missing value is represented by a single question mark. Values of string and nominal attributes are case sensitive, and any that contain space or the comment-delimiter character % must be quoted.

Here is an example of the Heterogeneity Activity Recognition dataset in .arff format.



```
@relation Giro_Dtrain-weka.filters.unsupervised.attribute.NumericToNominal-Rlast

@attribute Arrival_Time numeric
@attribute Creation_Time numeric
@attribute x numeric
@attribute y numeric
@attribute z numeric
@attribute gt {-1,0,1,2,3,4,5}

@data
1424688536759,266655896702000,0.377209,0.069333,-0.821308,-1
1424779943706,1424779947189185280,0.13298,0.165604,0.288361,5
1424688537179,6941511883000,0.445018,0.079781,-0.77683,-1
1424697720489,275277486683000,0.774272,-0.153938,-1.982869,2
1424699124715,1424699122722286590,0.019882,0.005127,-0.007751,1
1424696706486,52606592083000,-0.047647,-0.021075,0.033598,4
1424777784272,132938062041000,0.080023,-0.003665,-0.177151,2
1424694934455,12874044228000,0.854903,-0.496498,0.904187,3
1424777010125,1424778855443726340,0.019577,0.023254,-0.378418,-1
1424776556648,1424778401965047810,0.008682,0.012863,-0.037033,4
1424687989910,6478762133000,-0.501133,-0.083685,-0.054407,3
1424699392111,55107632116000,-0.007941,0.012217,0.002138,1
1424779348226,1424781193544403710,-0.002853,0.006165,0.002899,1
1424778180115,344036076742000,0.212581,0.067501,-0.451429,-1
1424694816299,1424694814308395780,-0.483368,-0.418396,-0.231552,3
1424782703678,1424784548993542140,0.29985,-0.011734,-0.231857,3
```

Figure 21: Arff file structure example

3. Experimental design and set-up

This section contains the different aspects of the set-up used for the experiments made during the project including the how the code and data was organized, the compilation, how the different code was executed in both x86 and ARM machines and the tools used to retrieve the final data to be analyzed.

3.1 Filesystem structure

Algorithms are the main component of this research. Therefore, the filesystem structure was based on them separating the code from the rest of the elements. Since both Marenostrum4 and ARM machine use the same OS, it was possible to use the same structure for both systems but with certain changes in the ARM machine. The main directory's name is TFM_RESOURCES and it contains the following elements:

- LR: This directory contains the code and the elements necessary for the Linear Regression algorithm implementation with the following elements:
  - linearR.py
  - Other_linearR.
  - LinearRegression.java
  - LRLaunch_Py.sh
  - LRLaunch_java.sh
- LogR: This directory contains the code and the elements necessary for the Logistic Regression algorithm implementation with the following elements:
  - LR.py
  - Other_LR.py
  - LogisticRegression.java
  - LogRLaunch_Py.sh
  - LogRLaunch_java.sh
- DT: This directory contains the code and the elements necessary for the Decision Tree algorithm implementation with the following elements:
  - DecisionTree.py
  - Other_ DecisionTree.py
  - DecisionTree.java
  - DTLaunch_Py.sh
  - DTLaunch_java.sh
- RF: This directory contains the code and the elements necessary for the Random Forest algorithm implementation with the following elements:
  - RandomF.py
  - Other_ RandomF.py
  - MyRandomForest.java
  - RFLaunch_Py.sh
  - RFLaunch_java.sh
- NB: This directory contains the code and the elements necessary for the Naïve Bayes algorithm implementation with the following elements:

- NB.py
  - OtherNB.py
  - NaiveB.java
  - NBLaunch_Py.sh
  - NBLaunch_java.sh
- K_Nearest: This directory contains the code and the elements necessary for the K-Nearest Neightbour algorithm implementation with the following elements:
  - K_Nearest.py
  - Other_K_Nearest.py
  - KNN.java
  - KNNLaunch_Py.sh
  - KNNLaunch_java.sh
- SVM: This directory contains the code and the elements necessary for the Support Vector Machine algorithm implementation with the following elements:
  - SVM.py
  - Other_SVM.py
  - CSV2Arff.java
  - SVMLaunch_Py.sh
  - SVMLaunch_java.sh
- WEKA: This directory contains the necessary libraries for the implementation and execution of the java version of the algorithms:
  - weka.jar » Main Weka library which is needed in all the java implementations.
  - mtj.jar » Additional Weka library which consists of a collection of matrix data structures, linear solvers, least squares methods, eigenvalue, and singular value decompositions. This one is needed for the Linear Regression implementation.
  - libsvm.jar » Additional Weka library needed to implement the Java version of the Support Vector Machine algorithm.
- Datasets:
  - Giro_Dtest.csv » Test dataset from the Heterogeneity Activity Recognition dataset for the Python version.
  - Giro_Dtrain.csv » Training dataset from the Heterogeneity Activity Recognition dataset for the Python version.
  - Giro_Dtest.arff » Test dataset from the Heterogeneity Activity Recognition dataset for the Java version.
  - Giro_Dtrain.arff » Training dataset from the Heterogeneity Activity Recognition dataset for the Java version.
  - xor_Dtest.csv » Test dataset from the Physical Unclonable Functions dataset for the Python version.
  - xor_Dtrain.csv » Training dataset from the Physical Unclonable Functions dataset for the Python version.
  - javaXorTest.arff » Test dataset from the Physical Unclonable Functions dataset for the Java version.
  - javaXorTrain.arff » Training dataset from the Physical Unclonable Functions dataset for the Java version.
  - productD_test.csv » Test dataset from the Product Analytics dataset for the Python version.
  - productD_train.csv » Training dataset from the Product Analytics dataset for the Python version.

  - productD_test.arff » Test dataset from the Product Analytics dataset for the Java version.
  - productD_train.arff » Training dataset from the Product Analytics dataset for the Java version.
  - CSV2Arff.java » This program is used to transform the .csv files into .arff files used in the Java implementation with Weka.

- CompileAll.sh: This script is used to prepare the whole set-up. It loads the necessary modules depending on if this is for the MareNostrum4 or ARM machine and compiles every java code file according to the necessary libraries that each implementation needs to be linked.
- LaunchJobs.sh: This script is used to launch all the necessary jobs in their own batch file.
- DataCollector.sh: This script is the one responsible to collect the data from the previous executions and write the results in a text file for the analysis.

Apart from the elements above, each algorithm directory also contains its own implementation of a program called Splitter.py. This program is used to do the preprocessing of the datasets according to the characteristic of the specific algorithm including the elimination of useless attributes and filling the empty spaces. It also splits the datasets in the training and testing datasets in a 80/20 proportion, as explained before.

3.2 Compilation

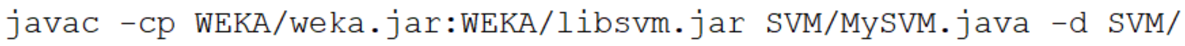
The compilation process of the whole system depends on the programing language. In the case of Python, as it was explained before, the compilation is made by the interpreter and does not generate a final like Java or other compiled programing language do. In the standard installation of the python system, the “python” command internally compiles and executes the code so it doesn’t require a previous step to be executed.

Java on the other hand require to be compiled before being executed since the execution requires that the .class file generated after the compilation process exists in the system. Due to the nature of the MareNostrum software environment it is not possible to install the required libraries in the system and add them to the classpath of the java subsystem. Therefore, the compilation requires to link dynamically the additional libraries. Also thanks to the java portability this could be replicated in the ARM machine.

The compilation command in the java jdk is javac. As it was stated before, the compilation requires the libraries to be linked dynamically and this can be made by adding the ‘-cp’ or ‘classpath’ option to the compilation command and the path of the needed libraries. Every java implementation of the algorithm requires the Weka library as it is the selected library for this project. The compilation command would then be like the following one:

```
javac -cp WEKA/weka.jar LR/LogisticRegression.java -d LogR/
```

As the java implementation of both Linear regression and Support Vector Machine algorithms require extra libraries, they can still be added in a similar way as the previous one like this:



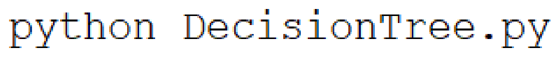
```
javac -cp WEKA/weka.jar:WEKA/libsvm.jar SVM/MySVM.java -d SVM/
```

It must be noted that Weka posses its own package manager system to easily install and use these extra libraries but for the same reason the link was made manually.

This whole process was resumed in the ColmpileAll.sh script which allows the user to prepare the entire system to be executed and again thanks to java portability this could work in any machine as long as it has its own java virtual machine installed.

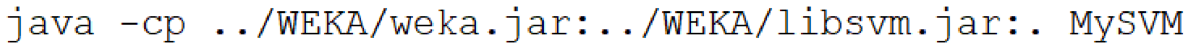
3.3 Execution

The execution command for every python program does not require special options to be added for the implementations made in this project so simply using the "python" command allows the user to compile and execute the code. An example would be the following:



```
python DecisionTree.py
```

In the case of java, it requires a similar solution to the compilation one, where the necessary uninstalled libraries need to be linked dynamically and manually using the '-cp' or '-classpath' option when using the execution command which in this case is 'java'. It must be noted that when some libraries are linked dynamically during a java program execution, the user must include the directory that contains the '.class' file obtained during the compilation process in the 'cp' option together with the path of the necessary libraries. An example of this would be the following:



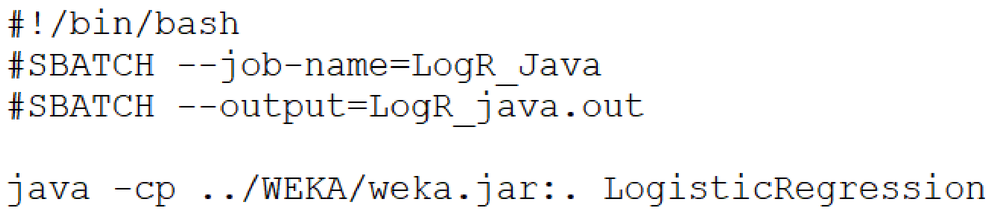
```
java -cp ../WEKA/weka.jar:../WEKA/libsvm.jar:. MySVM
```

In some cases, and due to the size of the datasets used, the execution code may exceed the heap size of the java virtual machine. When this happened the heap size for the execution of some java programs was changed by adding the '-Xmx' option to the execution command which allows the user to specify the heap size of the java virtual machine for the execution of a specific program.

In the Filesystem structure section, it can be seen that each algorithm directory posses a script ended in 'Py' or 'java'. These are scripts were made to execute the corresponding version of the implementation of the algorithm separately in a different job.

The execution process differs depending on the nature of the used system as the MareNostrum x86 and ARM machines use different job schedulers.

As it was explained in previous sections, the job scheduler of the MareNostrum4 system is SLURM. This means that every job executed in this system must fulfil the syntax characteristics of the SLURM batch scripts. This includes the batch directives and the list of command the user wants to execute in the system. An example of one of these files would be the following:



```
#!/bin/bash
#SBATCH --job-name=LogR_Java
#SBATCH --output=LogR_java.out

java -cp ../WEKA/weka.jar:. LogisticRegression
```

In the image above only two SLURM directives where used:

- #SBATCH --job-name=LogR_Java: This identifies the job scheduled by its name and will be important later in the data collection phase.
- #SBATCH --output=LogR_java.out: This generates a file with the output of the program and it is only used for debugging purposes.

The ARM machine uses a different job scheduler than the x86 MareNostrum and has a peculiarity. This scheduler contains a filesystem specifically dedicated for executions which is /fefs/scratch. This directory possesses a directory for every group inside it and a subdirectory for every user in the group directory. This means that all the necessary scripts, input files, code and any other kind of data must be transferred to this filesystem before launching a job.

The structure of the batch files required for this scheduler as a similar structure than the SLURM ones. The only notable differences is that the directives start with #PJM instead of the #SBATCH from SLURM and the number of possible directives available for the ARM scheduler is much more reduced than SLURM due to the fact that it is still in development.

3.4 Data collection

The final step of the experiment set-up is the Data collection process where the performance data of the executions is retrieved to be analyzed. This process is also dependent on the characteristics of the system used.

In the x86 MareNostrum system and due to the characteristics of both Java and Python as high-level programming languages that has almost no access to the physical resources such as main memory, SLURM was used to collect the performance data of the implemented code. As it was explained before this is made via the sacct command which provides the user with a large variety of information about the execution time and resource usage of a specific job or a group of jobs.

In the previous section it was stated that each job was launched with a specific name. This was made to identify this job later, when the DataCollector.sh script takes the performance information of the desired jobs. The DataCollector.sh script uses the sacct command to retrieve the data of specific launched jobs identified by its name and prints the value in a 'Result.txt' file which will be used for the analysis.

The ARM machine does not provide this feature and therefore the most suitable way to retrieve this data is by using the Python code profiler for the Python implementation and the internal functions of the Java library to collect the information provided by the system itself about the resource usage although this may interfere with the performance of the garbage collector system and therefore with the performance of the entire program, so this need to be taken in count when looking at the results.

4. Key results obtained in the study

This section intends to show the results obtained during this project. This section is divided in two sections. The first one shows the results regarding the process of coding and implementation of the different algorithms in both languages including the API provided by the used libraries and the size of the resulting final code. The second section will analyze the performance obtained by the implemented code in both x86 and Arm architecture as well as the causes of the obtained results.

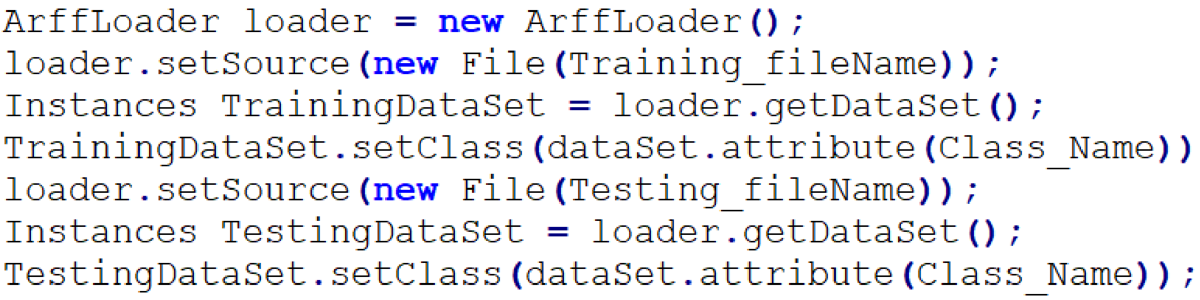
4.1 The code

When comparing two languages, code is an important factor to analyze. The syntax, the existing libraries and the simplicity of the expressions can make a huge difference when deciding which programming langue to use for a specific project.

In this project the implementation of the algorithms made use of existing AI libraries to provide a more realistic approach to a real implementation made by any researcher. This means that the code analysis is strongly related to the functionalities provided by these libraries.

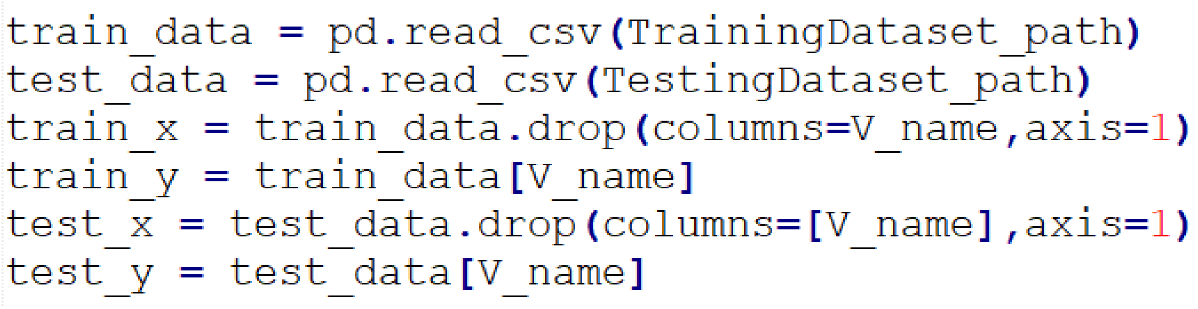
The coding process of an AI program normally requires the programmer to follow a series of well-defined steps.

The first one would be to load and manage the datasets that are going to be used. As for many other things, Java requires to create one or more objects to fulfil this task. In this specific case, the programmer first needs to create an ArffLoader object, provided by the Weka library, and then load the dataset. Then, a Instances object needs to be created to store the data from the dataset in the appropriate format and define the class attribute to be classified or predicted. Finally, repeat the process for the other dataset. The resulting code would be something similar to this:



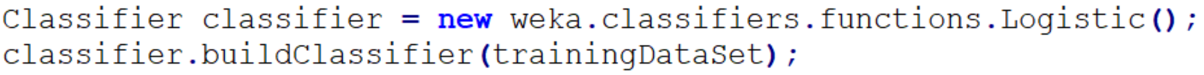
```
ArffLoader loader = new ArffLoader();
loader.setSource(new File(Training_fileName));
Instances TrainingDataSet = loader.getDataSet();
TrainingDataSet.setClass(dataSet.attribute(Class_Name))
loader.setSource(new File(Testing_fileName));
Instances TestingDataSet = loader.getDataSet();
TestingDataSet.setClass(dataSet.attribute(Class_Name));
```

Python on the other hand doesn’t need to declare the objects to be created, as it is a dynamically typed programming language. In this case, the programmer only requires calling the function provided by the Pandas library to load the datasets, pd.read_csv, and then separate the class attribute’s column from both training and testing datasets. The resulting code would be the following:



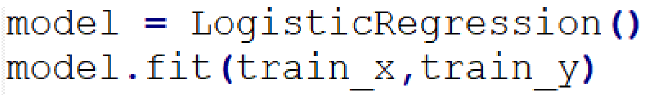
```
train_data = pd.read_csv(TrainingDataset_path)
test_data = pd.read_csv(TestingDataset_path)
train_x = train_data.drop(columns=V_name,axis=1)
train_y = train_data[V_name]
test_x = test_data.drop(columns=[V_name],axis=1)
test_y = test_data[V_name]
```

The next step would be to create the model of the algorithm a supply it with the training data. Again, Java requires to create an object to manage this which in this occasion is the Classifier object. Here the programmer can make use of the Java hierarchy system to assign a Classifier object an instance of any subclass of the Classifier class. In other words, we can assign any type of classifier to a Classifier object. In this case the required code is quite short. Here is an example:



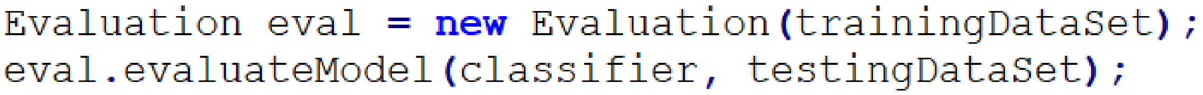
```
Classifier classifier = new weka.classifiers.functions.Logistic();
classifier.buildClassifier(trainingDataSet);
```

The python implementation of the model creation and training data feeding is almost identical to Java's. It only needs to create a variable to store the model and then feed it the training data.



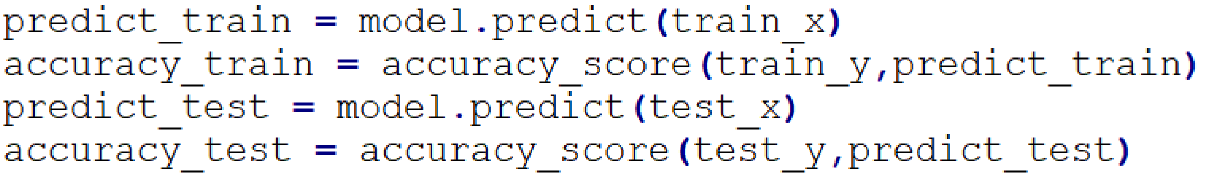
```
model = LogisticRegression()
model.fit(train_x,train_y)
```

The final step is to evaluate the model with the test data. The Weka library provides the user with the Evaluation class. This class contains a powerful method, evaluateModel(), which predicts the value of the previously defined class attribute and also provides a detailed description of the result obtained by the model compared to the testing data. Therefore, to implement this process in java the programmer only needs to create an object of the Evaluation class and call the evaluateModel function which again requires only a couple lines of code:



```
Evaluation eval = new Evaluation(trainingDataSet);
eval.evaluateModel(classifier, testingDataSet);
```

In this case Python requires the programmer to explicitly execute a prediction of the data and then compare it with the testing data. This produces the following code:



```
predict_train = model.predict(train_x)
accuracy_train = accuracy_score(train_y,predict_train)
predict_test = model.predict(test_x)
accuracy_test = accuracy_score(test_y,predict_test)
```

If one only takes a look at the previous process, the code required to implement a classifier for both Weka and Pandas/NumPy/Sklearn is almost identical in terms of code length and complexity. However, unlike Python, Java requires extra lines of code to create the complete Java source code file.

4.2 The performance

The performance result will be structured in 4 different sections. The first section will show the performance results of both Java and Python in the MareNostrum4 x86 architecture, the next two sections will focus on each programming language executed in both architectures and the final section will take a look at the results of Java and Python in the CTE-ARM machine. Every section will f show the results regarding the execution time and CPU efficiency first and then the memory usage.

4.2.1 Java and Python in x86

The analysis of the different tests and executions of the implemented programs for x86 show some interesting results. In the following table are showed the results of the execution time and the CPU efficiency:

Type of algorithm	Algorithm	Programming language	Elapsed time (s)	CPU time (s)	CPU efficiency (%)
Regression	Linear regression	Python	31.12	28.43	91.36
		Java	25.48	21.60	84.77
Classification	Logistic regression	Python	108.04	38.70	35.82
		Java	257.83	137.22	53.22
	Decision tree	Python	108.22	102.67	94.87
		Java	178.13	119.90	67.31
	Random Forest	Python	100.85	75.50	74.86
		Java	130.25	74.33	57.07
	K-Nearest	Python	158.40	132.77	83.82
		Java	180.88	143.29	79.22
	SVM	Python	197.50	178.18	90.22
		Java	240.43	211.34	87.90
	Naive Bayes	Python	33.32	19.30	57.92
		Java	53.74	29.61	55.10

Table 1: Execution time and CPU efficiency of Java and Python implementations in MareNsotrum4 Intel machine

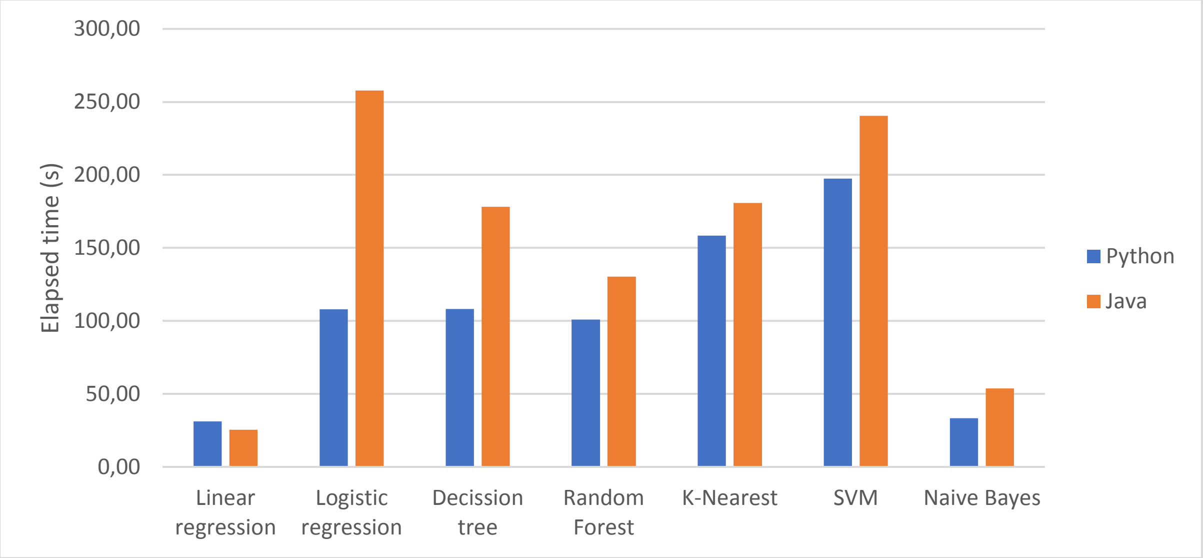


Figure 22: Graphical representation of Java and Python's execution time in MareNostrum4 Intel machine

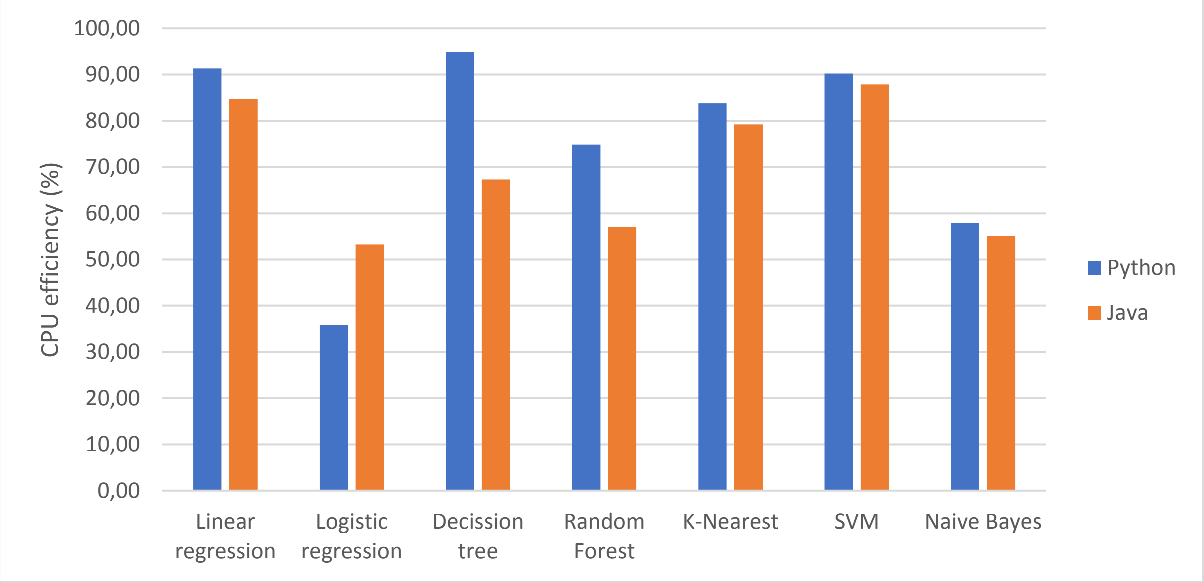


Figure 23: Graphical representation of Java and Python's CPU efficiency in MareNostrum4 Intel machine

Overall Java got a worst execution time in almost all cases with the exception of the Linear regression and Random Forest. This was a surprising outcome as other studies has showed that Java has in general better performance.

The most evident case is the Logistic Regression algorithm where the performance difference between Python and Java is significant. Although Java shows a better CPU efficiency, meaning that it spends more time computing data than doing other I/O tasks, the total elapsed time required to finish the execution is much higher. This is explained by the nature of how this algorithm is implemented in Weka and Scikit-learn. In Weka, the Logistic Regression class implementation is designed to create multinominal logistic regression models when the constructor is called. This means that the model expects that the class variable possesses 3 or more possible values and, therefore, the process to calculate the probability matrix includes extra computations that don't provide value to the final solution. On the other hand, Scikit-learn implementation automatically adapts the execution of the algorithm according to the nature of the input data which in the end provides a better performance (Weka, s.f.).

Decision Tree is also an interesting case. Although not as notable as the Logistic Regression case, the Java version takes almost 80% more time to finish its execution compared to Python. Again, the difference comes from the implementation of both algorithms in their respective libraries. Weka implements different Decision Tree algorithms and the most potent is the REPTree implementation (Weka, s.f.).

REPTree or Reduced-Error Pruning Tree is an algorithm that is specific to Weka. It is a fast decision tree learner that is optimised for simplicity and speed. The algorithms use reduced-error pruning with backfitting to find the smallest representation of the most accurate subtree with respect to the pruning set. Scikit-learn uses an optimized version of the CART algorithm or Classification and Regression Trees algorithm (Pedregosa, et al., 2011) although there is no information about the specific optimizations made by Scikit-lean. Weka barely provides any technical details of REPTrees. Thus, it's quite tricky to draw an exact line between REPTree and CART. The properties that are provided by Weka usually appear to be very similar to CART. More precisely, missing values and numeric attributes are handled just like in C4.5 which is very similar to CART, but it differs in that it doesn't support numerical target variables. Furthermore, the minimum number of instances per leaf, the maximal tree depth and minimum training set variance for a split can be specified. If we add this to the fact that Python has a better CPU efficiency it is possible to assume that the optimization is memory based as Java spends a considerable amount of time doing I/O operation which results in a worse performance.

Type of algorithm	Algorithm	Programming language	Memory used (KB)	Relative memory efficiency %
Regression	Linear regression	Python	608	13.86
		Java	534	-12.17
Classification	Logistic regression	Python	1923116	-16.66
		Java	2307484	19.99
	Decision tree	Python	1098316	-27.87
		Java	1522788	38.65
	Random Forest	Python	1923860	-5.33
		Java	2032132	5.63
	K-Nearest	Python	1037454	-7.70
		Java	1124020	8.34
	SVM	Python	1002634	-3.06
		Java	1034292	3.16
	Naive Bayes	Python	1924036	-14.35
		Java	2246370	16.75

Table 2: Memory consumption of Java and Python implementations in MareNsotrum4 Intel machine

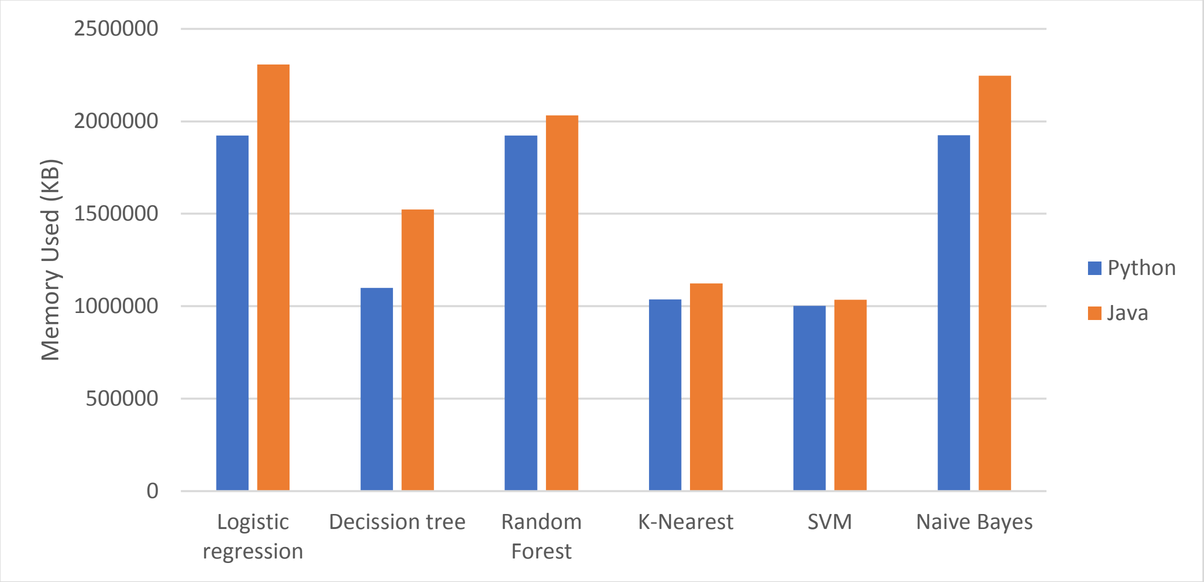


Figure 24: Graphical representation of Java and Python's memory consumption in MareNostrum4 Intel machine

The memory analysis of the performance provides the expected values. At stated before, previous studies show that Java performs bad regarding memory consumption and this case is no exception. Java requires more memory for every implementation and in some cases by a huge margin.

Looking at the Decision Tree algorithm, Java consumes almost 40% more memory than Python, which is quite impressive considering they both use the same data to execute the code and solve the same problem. The REPTree algorithm used in Weka, although is very accurate, it doesn't perform well when managing bif amounts of data.

4.2.2 Java in x86 and ARM

This section shows only the Java results in both machines and explain how the performance is affected by it.

Type of algorithm	Algorithm	Architecture	Elapsed time (s)	CPU time (s)	CPU efficiency (%)
Regression	Linear regression	x86	25.48	21.60	84.77
		ARM	23.01	20.13	87.48
Classification	Logistic regression	x86	257.83	137.22	53.22
		ARM	221.52	135.05	60.97
	Decision tree	x86	178.13	119.90	67.31
		ARM	184.90	121.98	65.97
	Random Forest	x86	130.25	74.33	57.07
		ARM	111.66	61.90	55.44
	K-Nearest	x86	180.88	143.29	79.22
		ARM	172.88	130.29	75.36
	SVM	x86	240.43	211.34	87.90
		ARM	201.30	182.93	90.87
	Naive Bayes	x86	53.74	29.61	55.10
		ARM	50.81	28.23	55.56

Table 3: Java's execution time and CPU efficiency in both MareNsotrum4 Intel and CTE-ARM machines

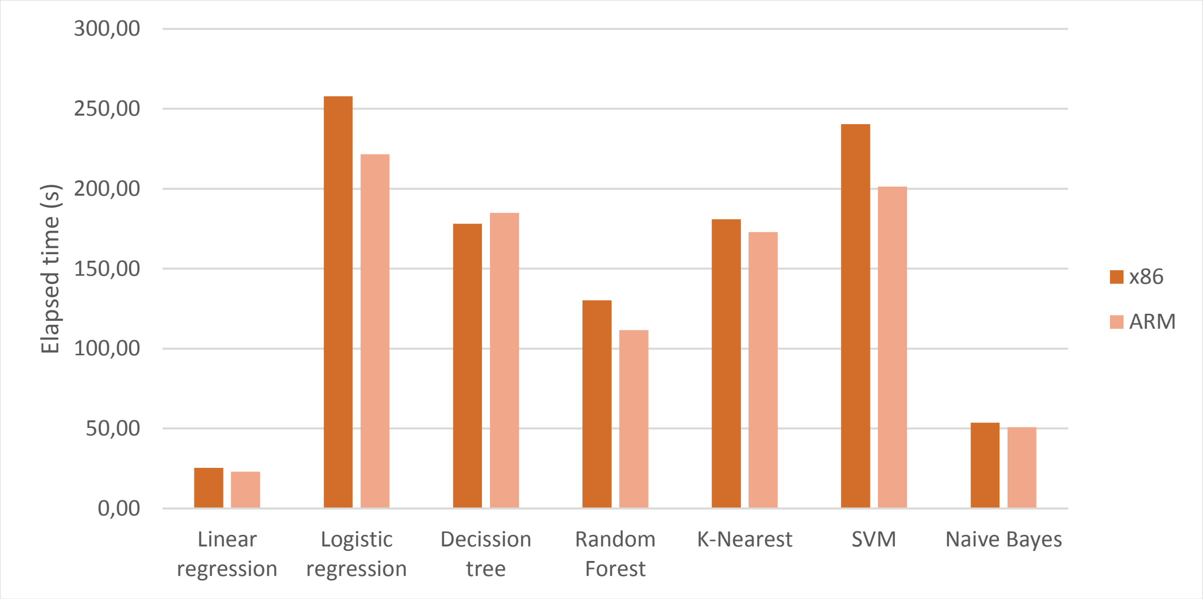


Figure 25: Graphical representation of Java's execution time in both MareNostrum4 Intel and CTE-ARM machines

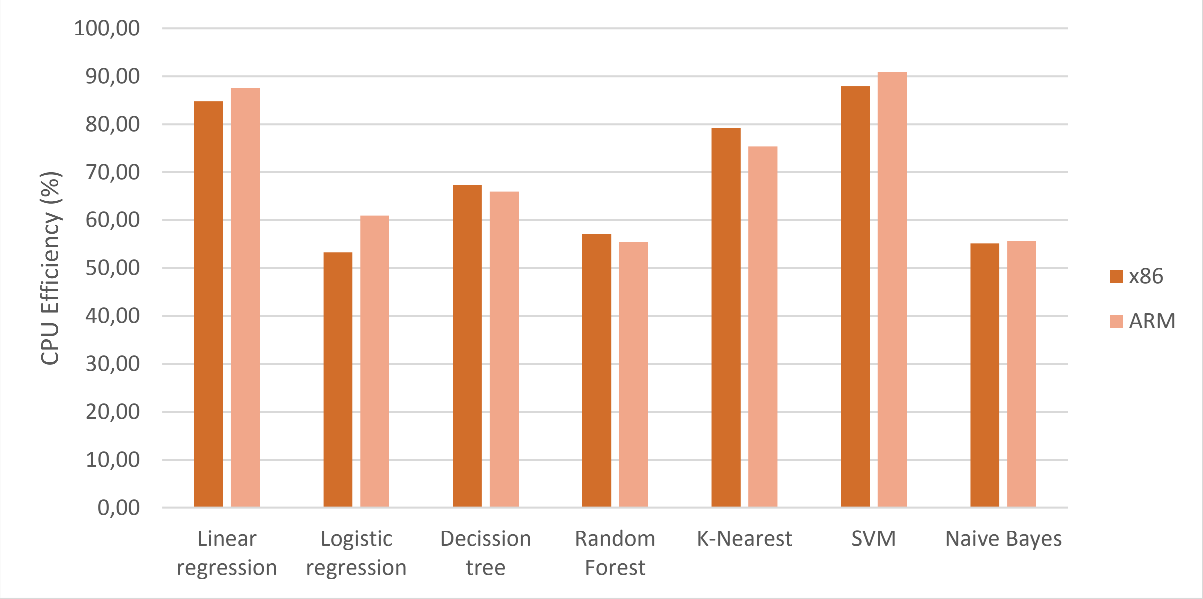


Figure 26: Graphical representation of Java's CPU efficiency in both MareNostrum4 Intel and CTE-ARM machines

These results show that the ARM architecture did affect the performance of the Java implementations. Overall ARM produced better execution times while having a similar CPU efficiency than x86 as shown in the table above.

This difference is explained by the specific hardware characteristics of both machines. The x86 performance result showed that Java obtained a worse performance mainly due to Java's memory management system as its CPU efficiency was lower in almost all the tested algorithms and the amount of the memory used was higher. As stated in the section 2.2 of this document, the x86 machine uses DDR4 RAM as main memory while the ARM machine possesses the HBM2 memory. Previous studies (Li, et al., 2018) have proven that the HBM memory can reduce end-to-end application execution time by 2-3x over DDRx architecture thanks to its faster and bigger bandwidth. The result is that java improves its elapsed time while maintaining a similar CPU time which implies that the performance increase is due to an improvement in the I/O process.

Type of algorithm	Algorithm	Architecture	Memory used (KB)	Relative memory efficiency %
Regression	Linear regression	x86	534	-12.46
		ARM	610	14.23
Classification	Logistic regression	x86	2307484	-3.27
		ARM	2385470	3.38
	Decision tree	x86	1522788	-8.17
		ARM	1658274	8.90
	Random Forest	x86	2032132	-4.03
		ARM	2117422	4.20
	K-Nearest	x86	1124020	-5.57
		ARM	1190274	5.89
	SVM	x86	1034292	-8.37
		ARM	1128752	9.13
	Naive Bayes	x86	2246370	-3.58
		ARM	2329862	3.72

Table 4: Java's memory consumption in both MareNsotrum4 Intel and CTE-ARM machines

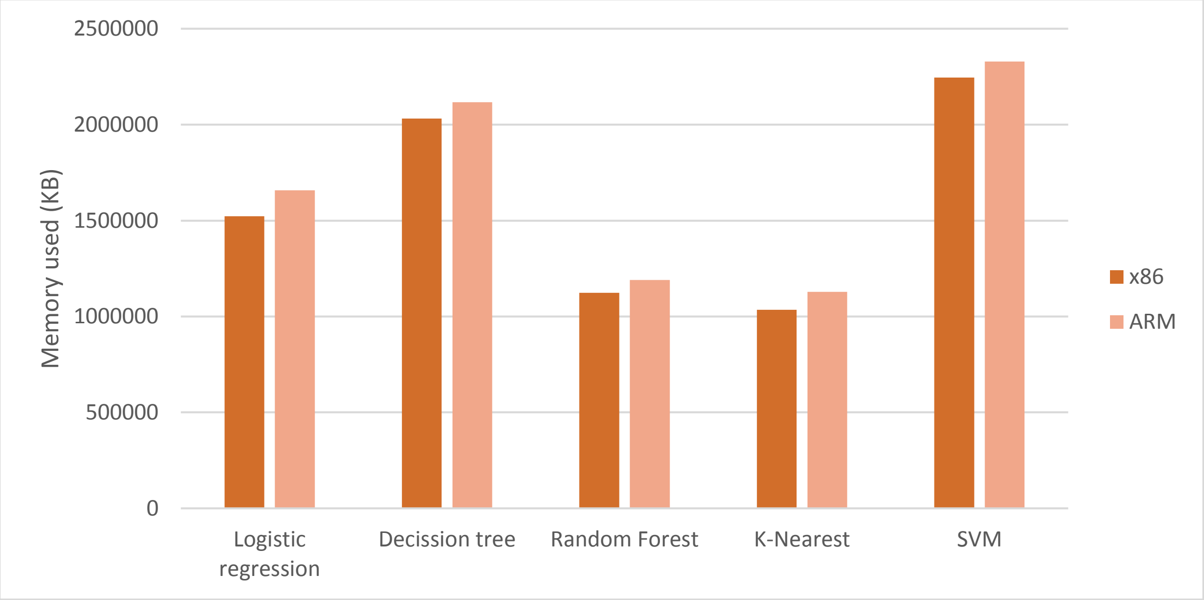


Figure 27: Graphical representation of Java's memory consumption in both MareNostrum4 Intel and CTE-ARM machines excluding Linear regression

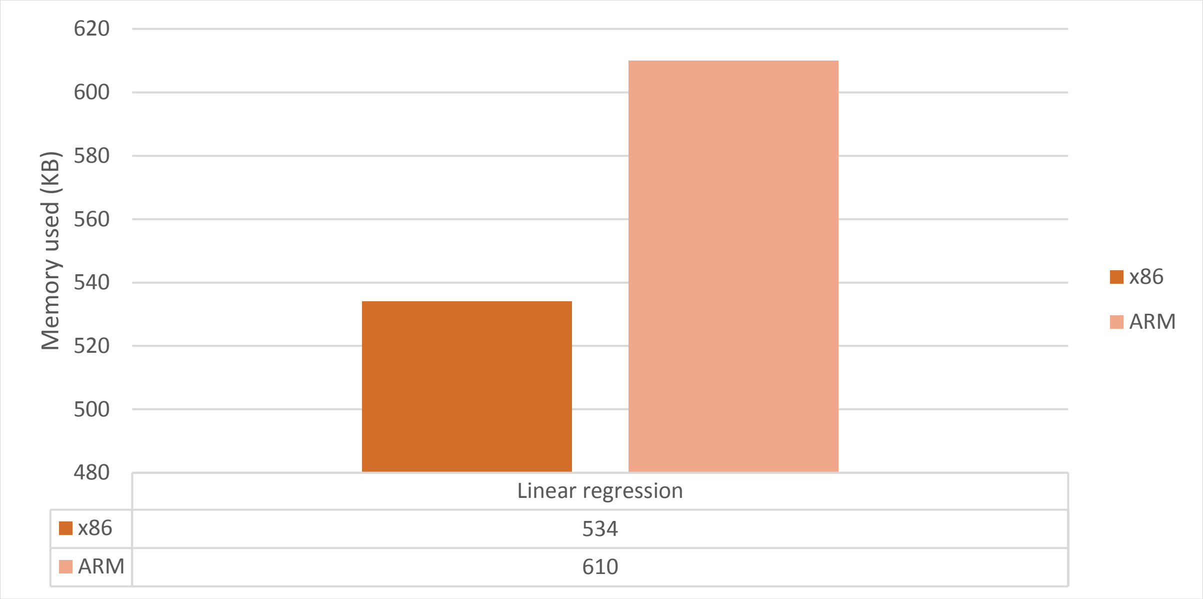


Figure 28: Graphical representation of Java's Linear regression memory consumption in both MareNostrum4 Intel and CTE-ARM machines

The only noticeable result regarding the memory usage for Java in both architectures is the slightly higher amount of memory required by the ARM implementations in every algorithm. No relation was found between this and the hardware properties or the specific implementation of each programming language.

4.2.3 Python in x86 and ARM

This section will show the results obtained for the Python implementations in both architectures.

Type of algorithm	Algorithm	Architecture	Elapsed time (s)	CPU time (s)	CPU efficiency (%)
Regression	Linear regression	x86	31.12	28.43	91.36
		ARM	23.86	21.69	90.91
Classification	Naive Bayes	x86	33.32	19.30	57.92
		ARM	28.57	15.87	55.56

Table 5: Python's execution time and CPU efficiency in both MareNsotrum4 Intel and CTE-ARM machines

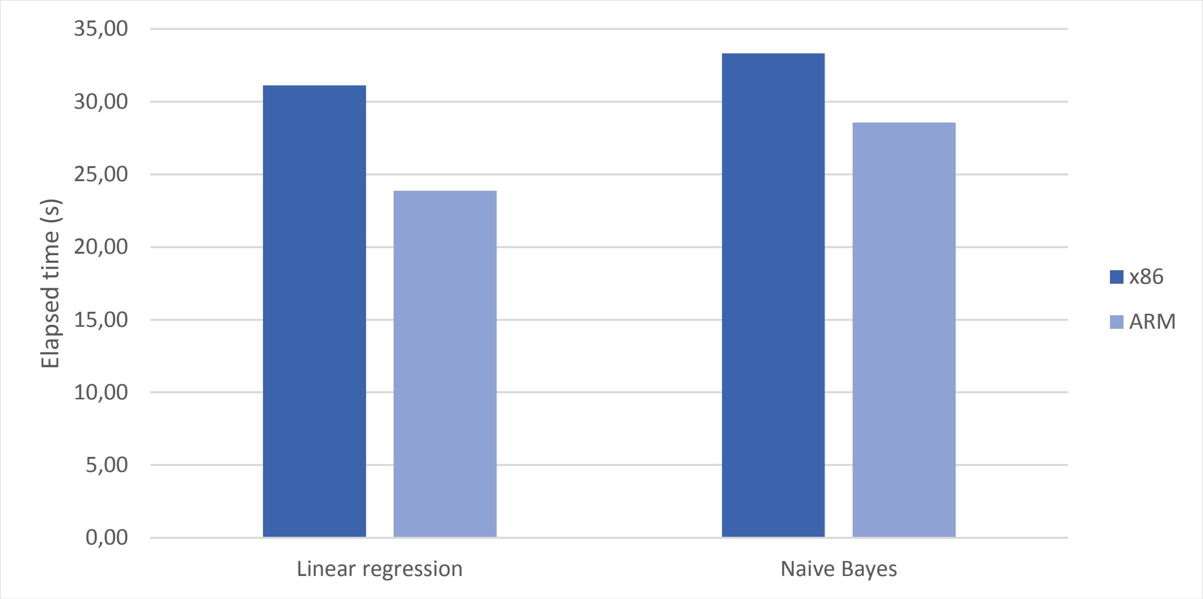


Figure 29: Graphical representation of Python's Execution time in both MareNostrum4 Intel and CTE-ARM machines

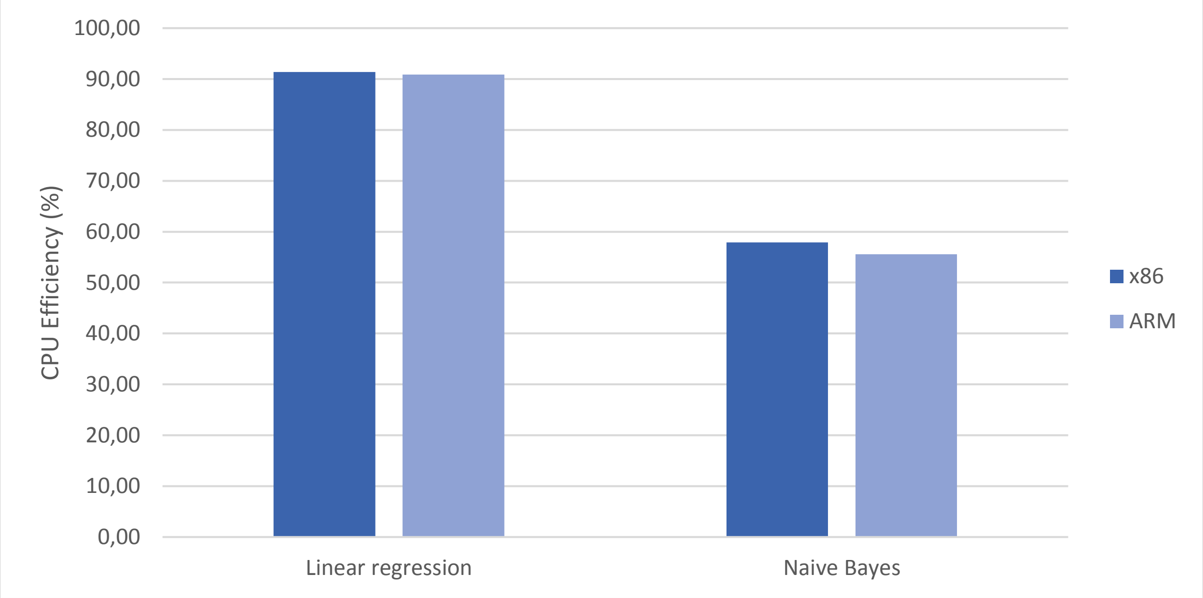


Figure 30: Graphical representation of Python's CPU efficiency in both MareNostrum4 Intel and CTE-ARM machines

As it is shown in the table above, Python has obtained a better performance in the ARM machine in both Linear regression and Naïve Bayes where it is apparent that this improvement comes from the reduced CPU time required by ARM. This means that ARM required less computation time to finish the execution. To explain this result, it is important to note the hardware difference, especially in the CPU used in both machines. As stated in section 2.2, the CTE-ARM machine uses a A64FX CPU designed by Fujitsu while the MareNostrum4 intel machine contains an Intel Xeon CPU. Recent studies have tested both processors performance (Jackson, et al., 2020) and the A64FX processor possesses a better one-core performance than the Xeon model used in the x86 machine. This explains the worse performance of the MareNostrum4 Intel machine against the ARM counterpart although the difference is not significant.

Type of algorithm	Algorithm	Architecture	Memory used (KB)	Relative memory efficiency %
Regression	Linear regression	x86	608	6.56
		ARM	650	-6.15
Classification	Naive Bayes	x86	1924036	-16.56
		ARM	1944036	19.85

Table 6: Python's memory consumption in both MareNsotrum4 Intel and CTE-ARM machines

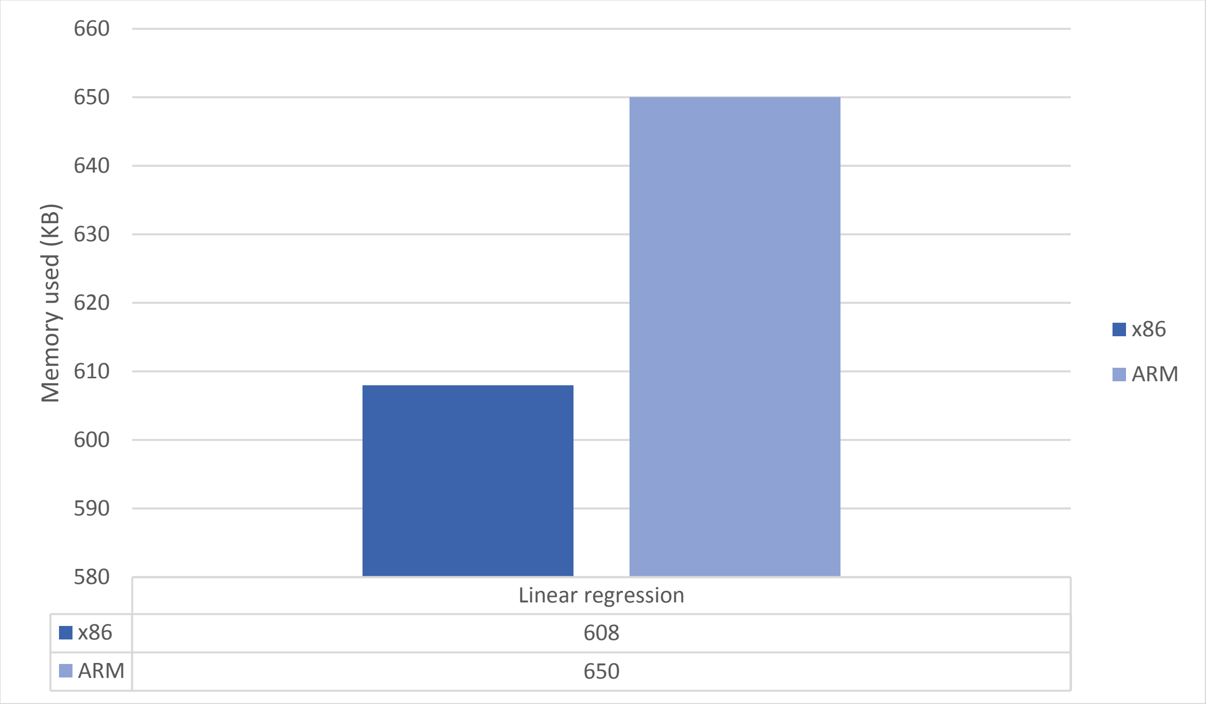


Figure 31: Graphical representation of Python's Linear regression memory consumption in both MareNostrum4 Intel and CTE-ARM machines

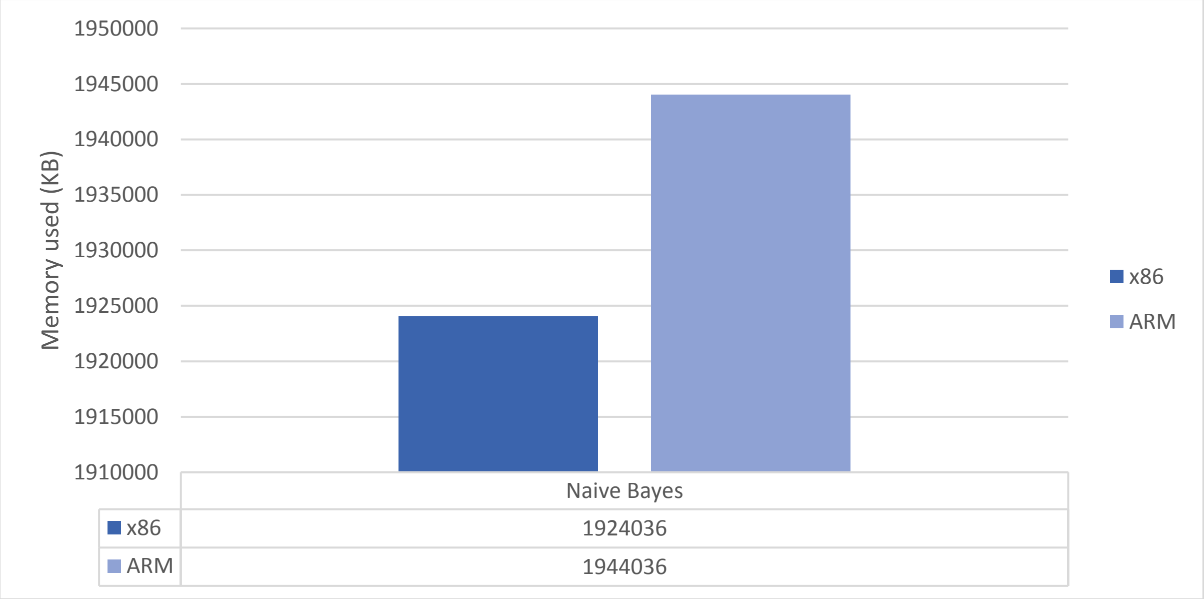


Figure 32: Graphical representation of Python's Naïve Bayes memory consumption in both MareNostrum4 Intel and CTE-ARM machines

4.2.4 Java and Python in ARM

This section shows the results obtained when analyzing the performance of Java and Python in the ARM architecture. Due to the limitations explained before, only two algorithms were able to be implemented in Python, The Linear Regression and Naïve Bayes. Therefore, these were the algorithms analyzed in this case.

Type of algorithm	Algorithm	Programming language	Elapsed time (s)	CPU time (s)	CPU efficiency (%)
Regression	Linear regression	Python	23.86	21.69	90.91
		Java	23.01	20.13	87.48
Classification	Naive Bayes	Python	28.57	15.87	55.56
		Java	50.81	28.23	55.56

Table 7:

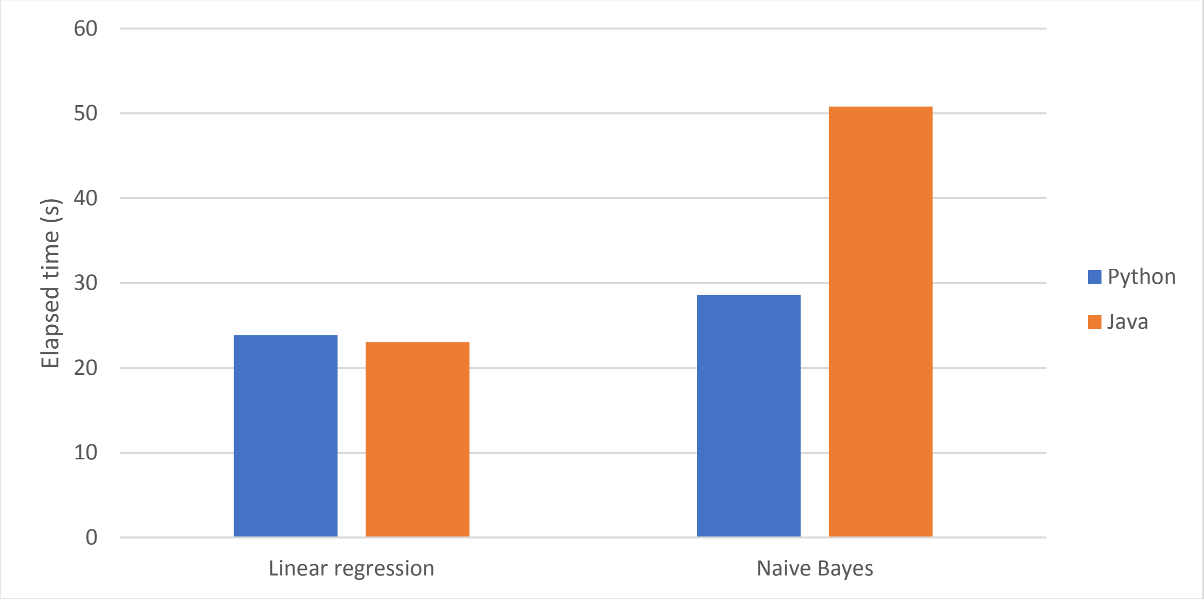


Figure 33: Graphical representation of Java and Python's execution time in MareNostrum4 CTE-ARM machine

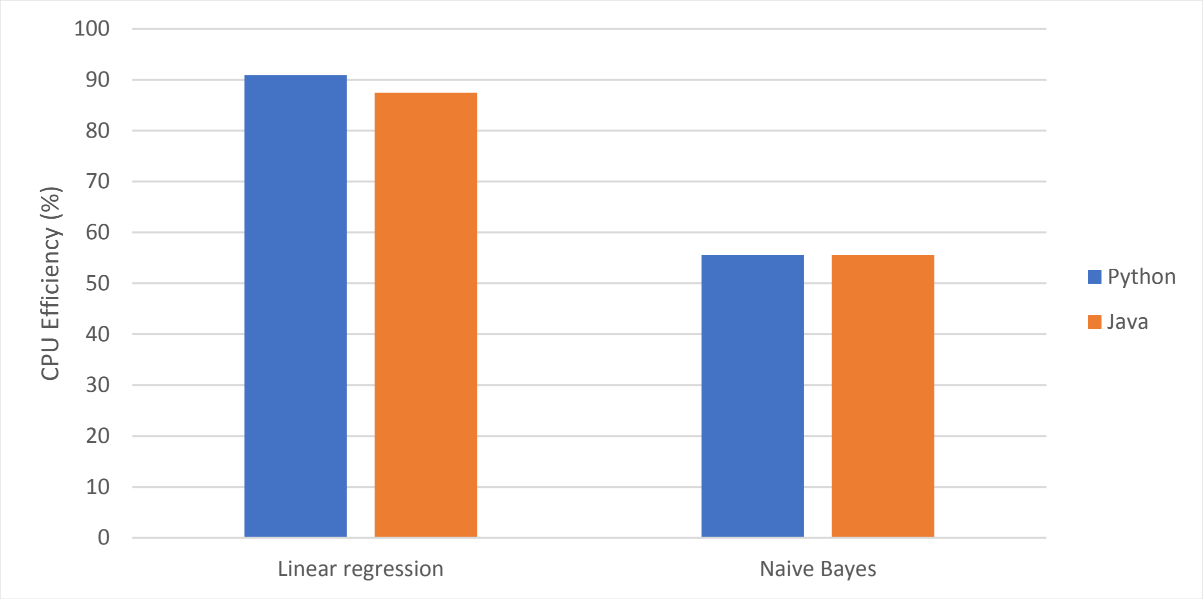


Figure 34: Graphical representation of Java and Python's CPU efficiency in MareNostrum4 CTE-ARM machine

The Java and Python performance in the ARM architecture only provides slightly different results than the x86. In that section, Java only obtained a better execution time in the Linear Regression algorithm and the same happens here although the difference is negligible. However, the existing differences between both implementations are bigger here.

Type of algorithm	Algorithm	Programming language	Memory used (KB)	Relative memory efficiency %
Regression	Linear regression	Python	650	6.56
		Java	610	-6.15
Classification	Naive Bayes	Python	1944036	-16.56
		Java	2329862	19.85

Table 8

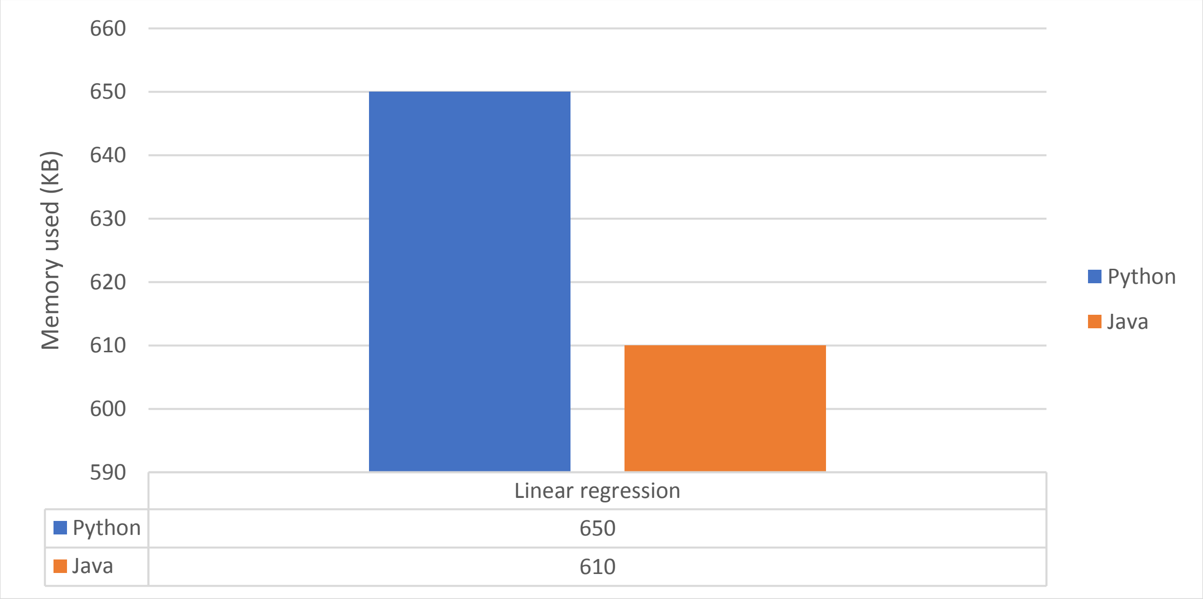


Figure 35: Graphical representation of Java and Python's Linear regression memory consumption in MareNostrum4 CTE-ARM machine

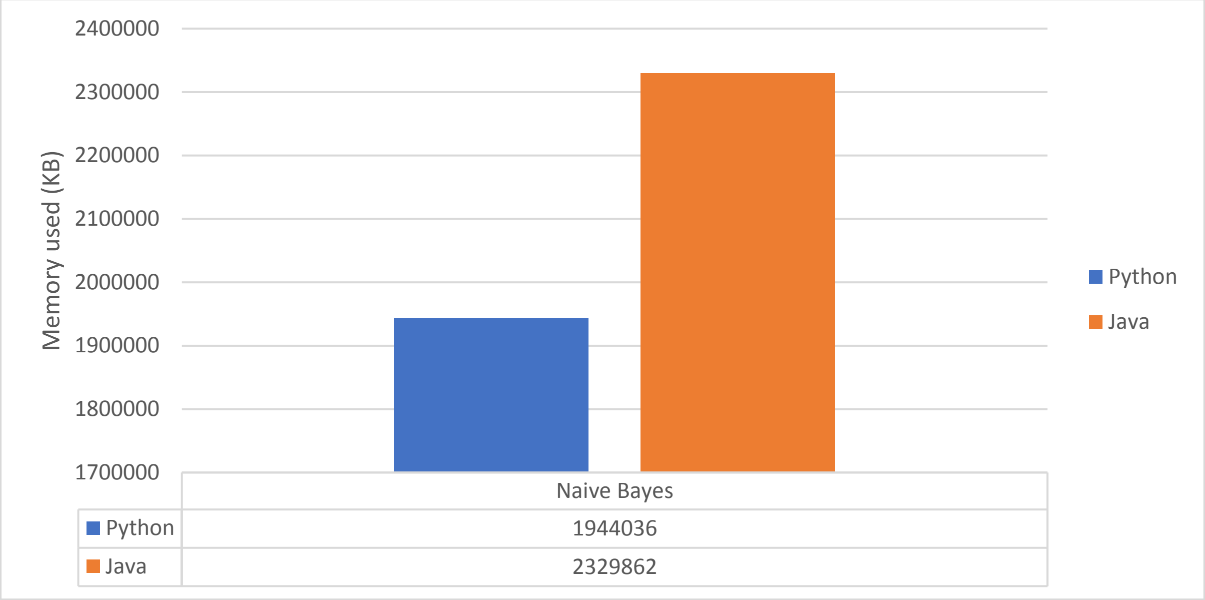


Figure 36: Graphical representation of Java and Python's Naïve Bayes memory consumption in MareNostrum4 CTE-ARM machine

5. Conclusions

This chapter is devoted to outline the principal conclusions obtained after the realization of the master thesis.

Both Java and Python have proven their strengths and weaknesses during the elaboration of this project. Even when using external libraries, which simplifies the necessary code to implement the desired solutions, Python has proven that it is still a more suitable programming language code-wise than Java to develop AI applications. Its simplicity and easy to use code provide a less time-consuming solution than Java, which still needs to implement the class structure and main method for any program coded in Java.

Regarding the performance, Java obtained some expected results in the memory consumption section as it has always been one of its weakest points as a programming language. The garbage collector system and the JIT compiler increase the memory consumption, especially when using large amounts of data which, given the nature and the size of the datasets used in this thesis has proven this to be true.

However, the execution time and CPU efficiency results obtained by Java in comparison with Python was a shocker. Although Java has always proven to be more memory consuming than other programming languages, its performance has been in general better than some of them, including Python. Here is where the importance of the code implementation importance comes to play. Weka is a relatively old programming library which, although it is still the most used Java programming library for developing AI programs, it is not updated as frequently as other, more modern libraries such as Scikit-learn and its getting outdated.

The inclusion of an ARM machine in this project adds more value to the results obtained in this thesis as it could be tested how much the architecture affect performance of programs written in high level programming language and more specifically, for AI programs. Overall, the executions made in the CTE-ARM machine provided better results in both Java and Python implementations. This was in part due to the better performance of Fujitsu’s A64FX one-core processing power and the more efficient memory hardware available in the CTE-ARM machine.

Finally, and given all the results obtained in this thesis is hard to consider that Java could nowadays outstand Python in the AI research area as a better programming tool. The performance provided by Weka implementations are not enough to compensate the extra work that requires to code the same algorithms in Java with respect to Python.

Future work

This section includes the future work that would be interesting to develop the project and provide more information that could lead to more solid conclusions and interesting results.

Multithreading

Future work includes implementing the current code using the multithreading characteristics provided by both languages and see if Java could provide more value in this area.

Using more features of the libraries provided algorithms

Now that some of the weak points of both java and Python have been detected, using the extra features of the implemented libraries to tune the executions could lead to some interesting results.

BSC tools

Tuning the existing BSC tools such as Extrae and Paraver to get information from Java and Python's performance could provide information about more aspects of the execution such as page faults, and instruction's related metrics for a more detailed and complete performance analysis and to find more weaknesses, not only of the programming langue, but also from the libraries themselves.

Further study of the coding process

This could include surveying programmers with different backgrounds and experience to test the difficulty process of coding AI algorithms in Java using Weka and Python using the previously mentioned libraries to give a more solid and complete view of the coding aspect of the analysis.

Other programming languages

Include other programming languages in the analysis such as C/C++ as well as the impact of C based implementation of Python applications such as Cython which have proved to drastically improve Python application's performance.

References

Anon., n.d. *Top500.* [Online]
Available at: https://www.top500.org/lists/top500/2021/06/

Bonakdarpour, B., Navabpour, S. & Fischmeister, S., 2011. *Sampling-based Runtime Verification.* Limerick, Ireland, s.n.

Caia, X., Petter, H. & Moea, H., 2005. On the performance of the Python programming language for serial and parallel scientific computations.

Destefanis, G. et al., 2016. A Statistical Comparison of Java and Python Software Metric Properties.

Fries, A., 2012. *The use of Java in large scientific applications in HPC environments,* Barcelona: Universitat de Barcelona.

Georges, A., Buytaert, D. & Eeckhout, L., 2006. Statistically Rigorous Java Performance Evaluation.

Gocht, A., Schöne, R. & Frenzel, J., n.d. Advanced Python Performance Monitoring with Score-P.

IBM, n.d. *IBM.* [Online]
Available at: https://www.ibm.com/docs/en/platform-rtm/9.1.4?topic=faqs-how-is-cpu-efficiency-calculated

Jackson, A. et al., 2020. Investigating Applications on the A64FX.

Joshi, N., 2018. Hands-On Artificial Intelligence with Java for Beginners.

Khoirom, S. et al., 2020. *Comparative Analysis of Python and Java for Beginners.* s.l.:s.n.

Kleinbaum, D. G. & Klein, M., 2002. *Logistic Regression A Self-Learning Text Second Edition,* s.l.: Springer.

Li, S., Reddy, D. & Jacob, B., 2018. A Performance & Power Comparison of Modern High-Speed DRAM Architectures.

Pedregosa, F. et al., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research.*

Prechel, L., n.d. *Are Scripting Languages Any Good? A validation of Perl, Python, Rexx and Tcl against C, C++.* s.l.:s.n.

Schmidt, A. G., Weisz, G. & French, M., 2017. *Evaluating Rapid Application Development with Python for Heterogeneous Processor-based FPGAs,* s.l.: s.n.

van der Walt, S., 2011. The NumPy array: a structure for efficient numerical computation. *IEEE Computing in Science and Engineering.*

Voitylov, A., 2021. *Java Magazine.* [Online]
Available at: https://blogs.oracle.com/javamagazine/post/java-on-arm-processors-understanding-aarch64-vs-x86

Weka, n.d. *Weka Logistic Regression.* [Online]
Available at: https://weka.sourceforge.io/doc.dev/weka/classifiers/functions/Logistic.html

Weka, n.d. *Weka Rep Tree.* [Online]
Available at: https://weka.sourceforge.io/doc.dev/weka/classifiers/trees/REPTree.html

Wolf, M., 2014. *High-Performance Embedded Computing.* Second ed. s.l.:s.n.

Yan, X. & Su, X., n.d. *Linear Regression Analysis: Theory and Computing.* s.l.:s.n.