H2020 – ICT-13-2018-2019

Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

Grant No 824988

D4.7 Machine Learning Algorithms over Semi Honest Operation Modes – Final Version

May 21

Imprint

Contractual Date of Delivery to the EC: 31 May 2021

Author(s): **Ángel Navia-Vázquez (UC3M), Jesús Cid Sueiro (UC3M), Manuel Vázquez López (UC3M), Francisco González-Serrano (UC3M)**
Participant(s): **UC3M**
Reviewer(s): **Susanna Bonura (ENG)**
Giacomo Fecondo (FCA-ITEM)

Project: **Machine learning to augment shared knowledge in federated privacy-preserving scenarios (MUSKETEER)**

Work package: **WP4**
Dissemination level: Internal
Version: **6.0**

Contact: **angel.navia@uc3m.es**
Website: **www.MUSKETEER.eu**

Legal disclaimer

The project Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824988. The sole responsibility for the content of this publication lies with the authors.

Copyright

Executive Summary

This deliverable (D4.7 Machine Learning Algorithms over Semi Honest Operation Modes – Final Version) comprises the MUSKETEER Machine Learning Library (MMLL) needed to execute the distributed learning under POMs 4, 5 and 6, as well as some demonstration scripts to check the correct execution of the code. The list of available algorithms contains the models committed in this deliverable (Linear models (Linear Regression, Logistic Classifier, Ridge Regression), Kernel methods (Kernel Regression, Support Vector Machines) and Clustering (Kmeans)). The design and usage of the new models incorporated in this version of the library is analogous to the already available ones in the previous version, so the integration and use of the new methods do not present any special difficulty with respect to what is already integrated in the platform. MMLL has been successfully integrated in the MUSKETEER Client connector developed in WP7 and it is fully operative over the "pycloudmessenger" communication service implemented in WP3.

Document History

Version	Date	Status	Author	Comment
1	6 May 2021	For internal review	Angel Navia-Vázquez	First draft
2	10 May 2021	Internal review	Susanna Bonura	Review
3	10 May 2021	Internal review	Giacomo Fecondo	Review
4	12 May 2021	Addressing review comments	Angel Navia-Vázquez	
5	12 May 2021	Final review	Mark Purcell	
6	13 May 2021	Final	Gal Weiss	

Table of Contents

List of Figures

List of Acronyms and Abbreviations

Abbreviation	Definition
API	Application Programming Interface
AUC	Area Under (ROC) Curve
BSVM	Budget Support Vector Machine
CA	Consortium Agreement
CC	Client Connector
CN	Cryptonode
DC	Data Connector
DL	Deep Learning
FL	Federated Learning (a.k.a. FML)
FML	Federated Machine Learning (a.k.a. FL)
FR	Functional Requirements
GA	Grant Agreement
HBC	Honest But Curious (a.k.a SH)
KM	Kernel Method
KR	Kernel Regression
LM	Linear Model
LC	Logistic Classifier
LR	Linear Regression
ML	Machine Learning
MMLL	MUSKETEER Machine Learning Library
MLC	Multiclass Logistic Classifier
MN	Master Node
MBSVM	Multiclass Budget SVM
OS	Operating System
PERT	Program evaluation and review technique
POM	Privacy Operation Mode
PP	Privacy Preserving
PPML	Privacy Preserving Machine Learning (a.k.a. Privacy Preserving Data Mining)
ROC	Receiver Operating Characteristics
RR	Ridge Regression
SH	Semi Honest (a.k.a HBC)
SMC	Secure Multiparty Computation
UI	User Interface
WN	Worker Node

1 Introduction

1.1 Purpose

This deliverable comprises the final version of the Machine Learning Library covering POMs 4, 5 and 6, to be integrated/used in the MUSKETEER platform. From now on we will name this library as "MUSKETEER Machine Learning Library" (MMLL). Some demos to illustrate the behaviour of the MMLL are also provided. These demos are not intended to be a benchmark of the library, they are provided for illustration purposes (the complete benchmark will be carried out in the context of WP6). This deliverable will help other partners in the understanding and usage of the POM 4, 5 and 6 models and algorithms, to facilitate their integration and use in the MUSKETEER platform.

1.2 Related Documents

D4.7 is the deliverable associated to the task T4.4 (Algorithms over semi-honest privacy preserving operation modes), as indicated in the PERT diagram below. It uses as input previous outcomes of WP4 (D4.1, D4.2, D4.3, D4.6), where a preliminary version of the library design and usage has been described, as well as a possible usage in the form of a MUSKETEER demonstrator. It also takes as inputs the requirements and specifications detailed in WP2, and, although not indicated in the PERT diagram, it is also respectful with the functional requirements FR017-FR024 described in D3.1 in relationship with the communications library.

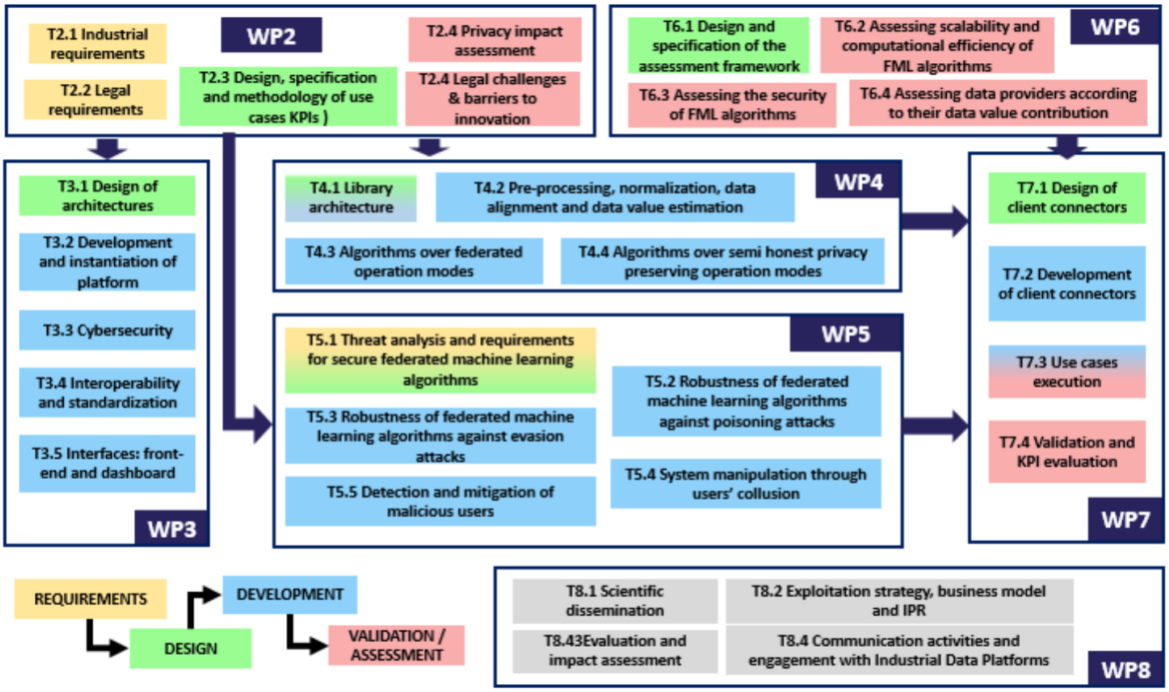

Figure 1 MUSKETEER's PERT diagram

1.3 Document Structure

This document is structured as follows:

- The current section (Introduction), presents the general aspects about this document and its relationship with other developments in the project.
- The section 2 ("Context of the Machine Learning Library") briefly revisits the main objectives of MUSKETEER from a Machine Learning point of view. We revisit some of the basic concepts about the platform execution, the participant processes, the corresponding objects and how they interact with other components of the platform such as the Client Connector or the Cloud Communications Library.
- In Section 3 ("POMs 4, 5 and 6 revisited") we also summarize the main principles underlying every Privacy Operation Mode (POM) and how the models under every POM will operate once integrated in the platform.
- The section 4 ("Methodology") describes the followed development process of the software components and the current state of the implemented models/algorithms delivered in MMLL v2.0.
- In Section 5 ("Library demonstration assumptions") we briefly describe the main steps needed to use the MMLL outside of the demos, the needed complementary information about the task context and the training data, and the contour conditions associated to the execution of training process in the context of POMs 4, 5 and 6.
- Section 6 ("MUSKETEER Machine Learning Library Setup and Usage") describes, on a step-by-step basis, the procedure to correctly install and execute the library in different Operating Systems (Windows, Linux and Mac OS).
- Section 7 ("MMLL Demos") provides the needed scripts to run the implemented models under every POM.
- Finally section 8 ("Conclusions") provides a summary on the contents of the deliverable and the obtained results.

2 Context of the Machine Learning Library

The library developed in this deliverable and described in this document is a fully operable implementation of the Machine Learning Library (MMLL) to be used in MUSKETEER under POMs 4, 5 and 6. Essentially, it aims at deploying a distributed ML setup (Figure 2b) such that a model equivalent to the one obtained in the centralized setup (Figure 2a) is obtained.

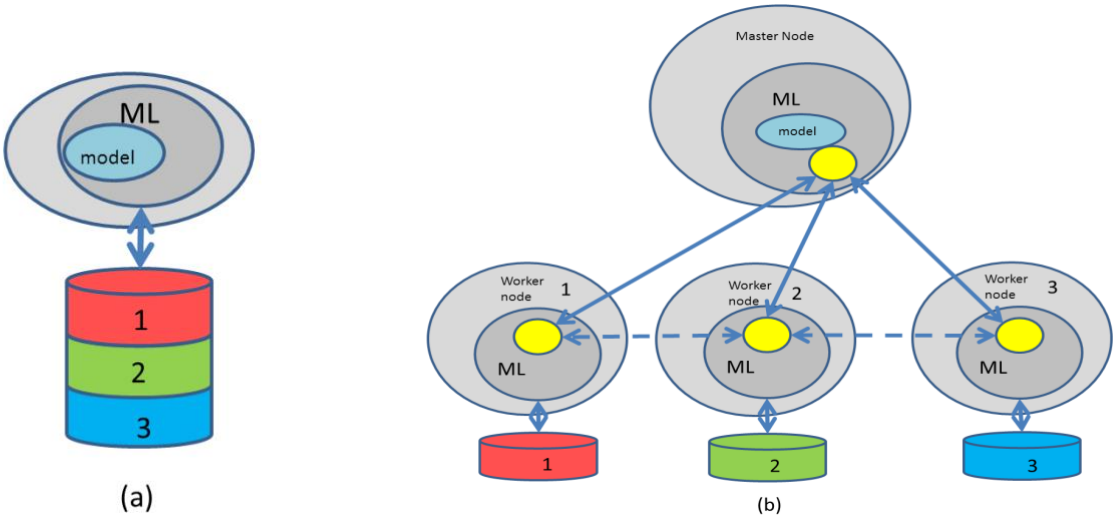

Figure 2 Centralized (a) vs. distributed scenario (b). Every user provides a portion of the training dataset. Data confidentiality must be preserved.

The centralized solution requires that the data from different users are gathered in a common location, something that is not always possible due to privacy/confidentiality restrictions. On the other hand, the distributed privacy preserving approach requires to distributedly compute some operation among the participating users such that a Master Node (MN) obtains the final ML model without ever receiving/seeing the raw data of the users and without revealing the trained model to the rest of participants (in POMs 4, 5 and 6).
In a second level of detail, we can describe the interaction among nodes as shown in the next Figure:

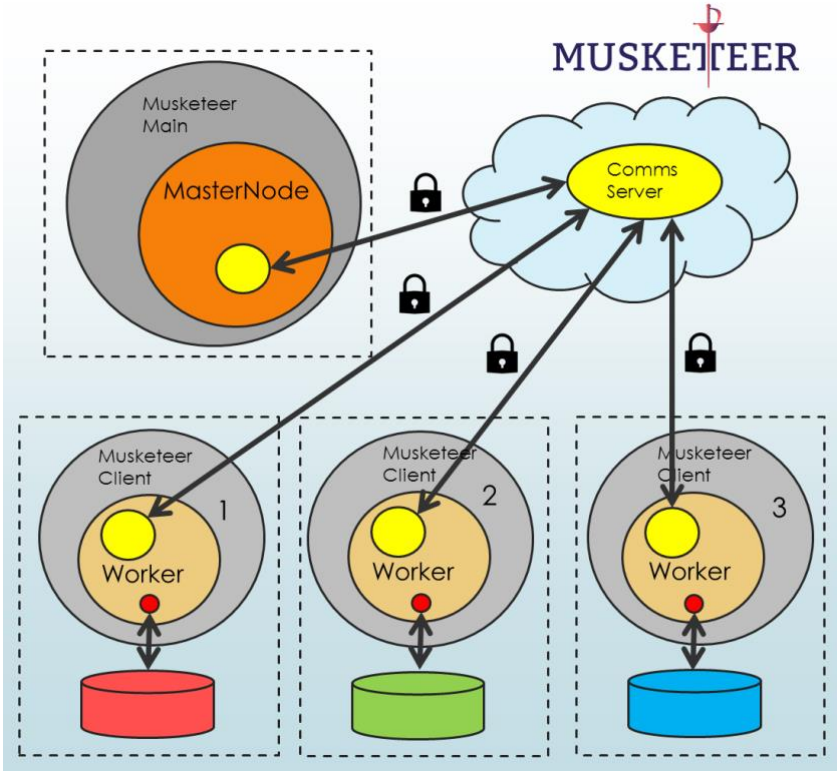

Figure 3 Detailed process interactions in a MUSKETEER learning process.

We observe the participation of several actors in a learning process, everyone marked as a dashed box and supposedly running on a different (remote) machine:

- The **MUSKETEER main process**: it is the process that orchestrates the training procedure, identifies the potential contributors and obtains the final model. It runs the "MasterNode" object (dark orange circle) from the MMLL. It communicates by means of the communication object (yellow circle) with the other participants through the Communications Service at the Cloud.
- The **MUSKETEER client**: it is the process that every participant must locally execute. It runs the "WorkerNode" object (light orange circle) from the MMLL. The Worker has access to the local data through the specific data connector (red circle) provided by the end user and communicates with the MN by means of the communication object (yellow circle) through the Communications Service at the Cloud.

In the next Section we describe in a deeper detail the structure of the objects participating in POMs 4, 5 and 6, as well as the expected interactions among the different types of nodes.

3 POMs 4, 5 and 6 revisited

General aspects:
The following nodes (objects) are to be executed:
Common to POMs 4, 5 and 6:

- **Master Node (MN)**: a central object (process) that controls the execution of the training procedure.
- **Worker Node (WN)**: an object to be executed in the end user side, possibly as a part of the MUSKETEER client. It is the only node that has a direct access to the raw data provided by every user, through an 'ad-hoc' Data Connector (DC).

Specific to POM 4:

- **Crypto Node (CN)**: an object providing some cryptographic services. It can be run anywhere but it cannot collude with the Master Node. It is only needed in POM 4, because POM6 does not use encryption and in POM5 the Master Node plays the role of Crypto Node.

In what follows we describe the normal operation of a training algorithm under every POM.

3.1 POM 4

This POM uses an additively homomorphic cryptosystem to protect the confidentiality of the data. The CN will help in some of the unsupported operations. The scheme is cryptographically secure if we guarantee that there is no collusion between the MN and the CN. In the next Figure we represent the interaction among the participants.

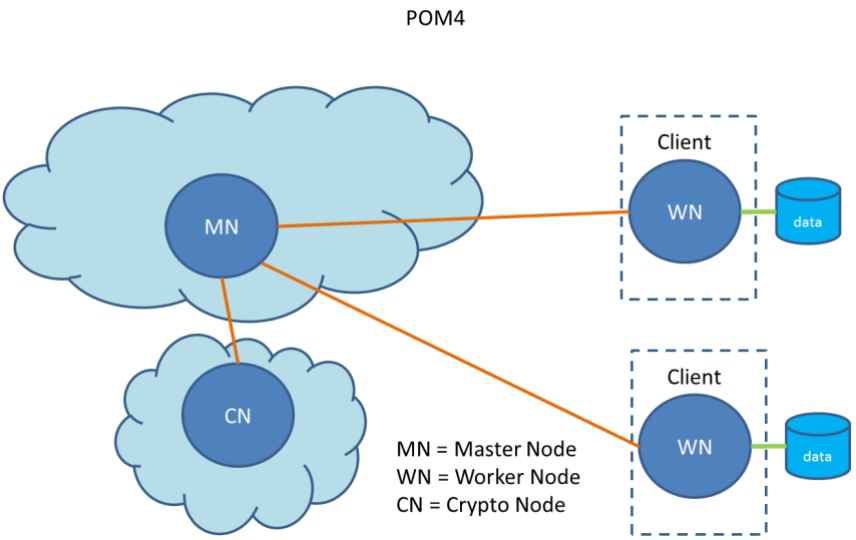

Figure 4 POM 4 general setup.

The steps to train a given model are:

1. The MN asks the CN to generate public and private keys. The public keys are distributed to the WNs. The CN keeps the private key.
2. Every WN encrypts the data with the public key and sends the encrypted data to the MN.
3. The MN starts the training procedure by operating on the model parameters and encrypted users data. The initial model parameters are generated at random by the MN.
4. The MN is able to perform some operations on the encrypted data (the homomorphically supported ones).
5. For the unsupported ones, it needs to establish a secure protocol with the CN consisting in:
    a. The MN sends some blinded data (masking) to the CN
    b. The CN decrypts the blinded data and computes the unsupported operation in clear text. Then it encrypts the result.
    c. The MN receives the encrypted result and removes the blinding.

    As a result of this protocol, the MN never sees the data or the result in clear text and the CN only sees the clear text of a blinded message, different from the raw data.
6. The procedure goes back to 5 until a stopping criterion is met.

POM 4 is a cryptographically secure procedure, providing that MN and CN do not collude.

3.2 POM 5

This POM uses an additively homomorphic cryptosystem to protect the confidentiality of the data and model. The data does not leave the participants facilities and the MN will play the role of a CN, by helping in some of the unsupported[1] operations. The scheme is cryptographically secure if MN and WN do not collude. In the next Figure we represent the interaction among the participants.

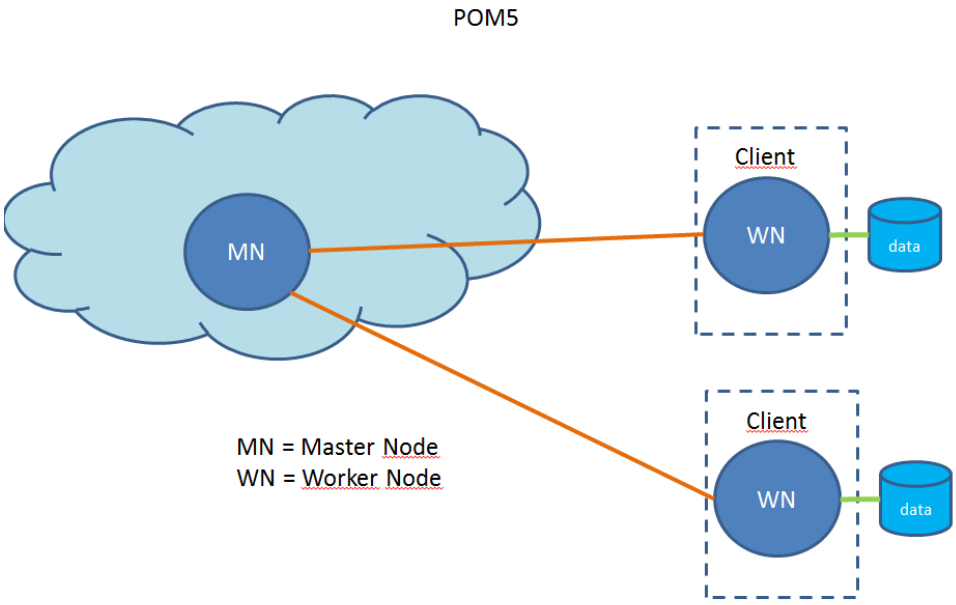

Figure 5 POM 5 general setup.

The steps to train a given model are:

1. The MN generates public and private keys. The public keys are distributed to all participants.
2. The initial model parameters are generated at random by the MN. The MN encrypts the model parameters with his secret keys and sends the encrypted model to the WNs.
3. The WN carry out the operations needed by the MN on the encrypted model and un-encrypted users data.
4. The WN is able to perform some operations on the encrypted data (the homomorphically supported ones).
5. For the unsupported ones, the WN needs to establish a secure protocol with the MN consisting in:
    a. The WN sends some blinded encrypted data (masking) to the MN

[1] In an additive homomorphic cryptosystem, like the one used here, the supported operations are the addition of two encrypted numbers and the multiplication of an encrypted number by a non-encrypted one. The non-supported operation is the multiplication of two encrypted numbers. In a multiplicative homomorphic cryptosystem, the unsupported operation is the addition of two encrypted numbers.

b. The MN decrypts the data and computes the unsupported operation in clear text. Then it encrypts the result.
c. The WN receives the encrypted result and removes the blinding.

As a result of this protocol, the MN never sees the data or the result in clear text, and the WN only sees the encrypted model.

6. The procedure goes back to 5 until a stopping criterion is met.

POM 5 is a cryptographically secure procedure whenever MN and WN do not collude.

3.3 POM 6

This POM does not use encryption; it relies on Secure Multiparty Computation and possibly other (two-party) Secret Sharing protocols to solve some operations on distributed data. In the next Figure we represent the interaction among the participants.

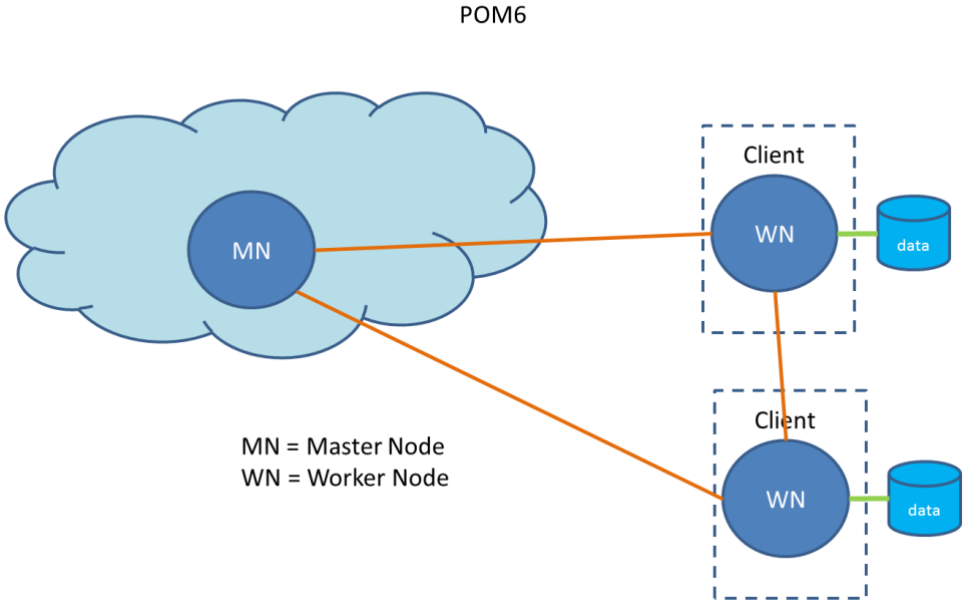

Figure 6 POM 6 general setup.

Under this POM, raw data is not encrypted, but it is never sent outside the WN. The model trained in the MN is also kept secret to the WN, since it never leaves the MN. Some transformations of the data can be exchanged with the MN, such as aggregated values, correlation matrices, etc. Every implemented algorithm will describe which information is revealed to the MN, for instance: covariance matrices, number of training patterns, average of the training patterns, etc. In any case, the raw data (individual training patterns) will not be revealed and cannot be obtained by inverse engineering on the exchanged data.
Some of the operations can be directly implemented using SMC protocols such as secure dot product, secure matrix multiplication, etc. The security of these operations will be as described in the reference sources describing every protocol. POM6 is not a general procedure, it requires that every algorithm is implemented from scratch, and it is not

guaranteed that any algorithm can be implemented under POM6. For some operations, a "round robin" protocol is required; therefore direct connections among some of the WNs are needed (ring network). The ring topology is already available at the "pycloudmessenger" communication service implemented in WP3.
As an illustrative example, let's imagine a training procedure that requires at every step to compute a model output using a dot product and receive the average covariance matrix among all the WNs. The procedure could be as follows:

1. The MN starts a SMC protocol to obtain the dot product with the data from every WN.
2. The MN asks the WNs to compute their covariance matrices.
3. The MN starts a round robin protocol with blinding to obtain the accumulated covariance matrix
4. Using the received information the MN updates the model (the specific correlation matrices of every worker are not revealed).
5. The procedure goes back to 1 until a stopping criterion is met.

4 Methodology

4.1 General development process

The library development followed these steps:

1.- Develop an algorithm prototype without communications library
2.- Adaptation to the provisional local communications library provided by IBM
3.- Preliminary version with the code structure agreed between UC3M and TREE
4.- MMLL preliminary version of the library (provided in Deliverable D4.6, as long as some demos)
5.- Integration check in the MUSKETEER Client Connector
6.- Integration with the final communications service (IBM Cloud)
7.- MMLL 2.0: final version of the library (provided in D4.7 (M30))

4.2 Current status of the library

We briefly describe here the current status of the algorithms in the context of POMs 4, 5 and 6:

Ridge Regression	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Ridge Regression DEMO v1.0.0	**Released (Provided in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Ridge Regression DEMO v2.0.0	**Released (Provided in D4.7)**

Linear Regression	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Linear Regression DEMO v1.0.0	**Released (Provided in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Linear Regression DEMO v2.0.0	**Released (Provided in D4.7)**

Logistic Classifier	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Logistic Classifier DEMO v1.0.0	**Released (Provided in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Logistic Classifier DEMO v2.0.0	**Released (Provided in D4.7)**

Multiclass Logistic Classifier	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Multiclass Logistic Classifier DEMO v1.0.0	**(Not available in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Multiclass Logistic Classifier DEMO v2.0.0	**Released (Provided in D4.7)**

Clustering (Kmeans)	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Clustering (Kmeans) DEMO v1.0.0	**Released (Provided in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Clustering (Kmeans) DEMO v2.0.0	**Released (Provided in D4.7)**

Kernel Regression	
1. Prototype without communications library	**Done**

2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Kernel Regression DEMO v1.0.0	**Released (Provided in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Kernel Regression DEMO v2.0.0	**Released (Provided in D4.7)**

Budget Support Vector Machine	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Budget Support Vector Machine DEMO v1.0.0	**(Not available in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Budget Support Vector Machine DEMO v2.0.0	**Released (Provided in D4.7)**

Multiclass Budget Support Vector Machine	
1. Prototype without communications library	**Done**
2. Adaptation to IBM's local communications library	**Done**
3. Preliminary version with common code structure	**Done**
4. Budget Support Vector Machine DEMO v1.0.0	**(Not available in D4.6)**
5. Algorithm & code optimization	**Done**
6. Integration with the final comms. service (IBM Cloud)	**Done**
7.- MMLL 2.0: final version of the library	**Released (Provided in D4.7)**
8. Multiclass Budget Support Vector Machine DEMO v2.0.0	**Released (Provided in D4.7)**

5 Library Demonstration assumptions

In what follows, we assume that a Machine Learning task has already been defined, either in the context of the Client Connector or using the specific scripts provided with MMLL-demo, and that the MUSKETEER platform has already identified all the potential users participating in the training process. In the end-to-end version of the MUSKETEER platform, the services allow users to register to the platform, define tasks, provide training data and join/manage tasks and resulting models, as described in the context of WP3/WP7.
Therefore, for the purpose of this demonstrator, we will assume the following:

- **General description of the task**: All participants have access to this description and agree to participate and contribute some data to the learning process. A preliminary check procedure has already been executed to guarantee that the contributed data follows the needed format (number and type of input features, number and type of

target values, etc.), as described in D4.3 (Pre-processing, normalization, data alignment and data value estimation algorithms – Final version).

- **User_addresses and execution**: all the participants have received the corresponding credentials to use the Cloud Communications Service and the list of addresses of the participating nodes (Worker Nodes (WN)) is available to the MasterNode, according to FR017 in D3.1, possibly after joining the corresponding task. Every participant (Master/Cryptonode/Workers) can be a separate process in a potentially different machine/location. The current "pycloudmessenger" version of the Communications Library (CL) is able to communicate between processes in different machines, through the IBM Cloud. All the reported experiments/demos will use this messaging service, no other direct communication between processes is used outside the data transmission protocols described in the context of WP3.
- **Data**: the data for training, validating and testing will be provided to MUSKETEER by means of a Data Connector (DC). For illustration purposes we provide in the MMLL-demo repository a DC to be used in the demonstrator that simply loads data from a file in a format suitable to execute these demos. The final DC for the user cases will be developed in the context of WP7 and any other user with special input/output requirements must provide a suitable DC. For the purpose of this demonstration we use well known public datasets.
- **Confidentiality requirements**: In the context of POMs 4, 5 and 6, we will assume that the raw data is never sent (in clear text, or unencrypted) outside of the owner's context and that the trained model is kept secret (only known to the Master Node). We will allow to exchange among the participants some transformations of the data (such as aggregations, cross-correlation matrices, encrypted values, etc.), but in any case that information cannot be used to reconstruct the raw input data or targets. The final end users will be aware in advance of the type of information exchanged under every Privacy Operation Mode (POM), and it is their ultimate responsibility to choose among one POM or another, taking into account that the extra confidentiality (model confidentiality) that POMs 4, 5 and 6 provide is at the cost of a computational overhead.
- **Communications library:** The MMLL is fully compatible with the Pycloudmessenger (currently v0.7) service developed in the context of WP3 and all the tests and experiments have been run using that service.

6 MUSKETEER Machine Learning Library Setup and Usage

This Section describes the needed steps to install the library and run the demo experiments on a variety of simulations to evaluate the correct operation of the library.

6.1 Software installation instructions

For the execution of the demos a correctly installed Python 3 environment has to be set up beforehand with all the dependencies correctly installed, including the MMLL library as well as the communications library used here (https://github.com/IBM/pycloudmessenger/) and any other dependencies. Although this information is present in the GitHub repository of the project (https://github.com/Musketeer-H2020/MMLL), the details of the configuration are going to be included also in this report.
The environment instructions are described for Anaconda (a Python distribution), although any other tool for managing Python 3 virtual environments could be used. First, we create a conda environment with the basic configuration and activate it:

```
conda create --name MMLL_demo python=3.7.4 git gmpy2==2.0.8 -c

    defaults -c conda-forge --yes

conda activate MMLL_demo
```

The final step is to install MMLL and its dependencies (including the "pycloudmessenger" library):

```
pip install git+https://github.com/Musketeer-H2020/MMLL.git

pip install http://github.com/IBM/pycloudmessenger/archive/v0.7.0.tar.gz
```

For a more complete description of the installation instructions, please refer to D4.6.

6.1 Software documentation

The MMLL library repository includes a detailed documentation of the comprised software modules. It can be accessed through the README file in the GitHub repository. We describe there the general module structure of MMLL, the main API functionalities (mainly accessible through the MasterNode, WorkerNode and CryptoNode components, ant the input/output definition of every implemented ML model/algorithm). In the next Figures we provide some illustrative samples of the documentation aspect, but the real one is in html format and has to be accessed through the provided link:

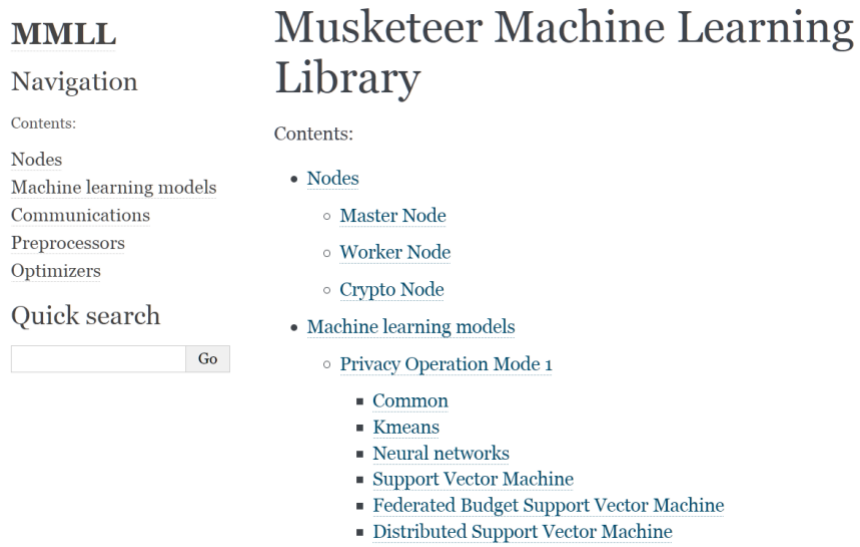

Figure 7 MMLL documentation: main page.

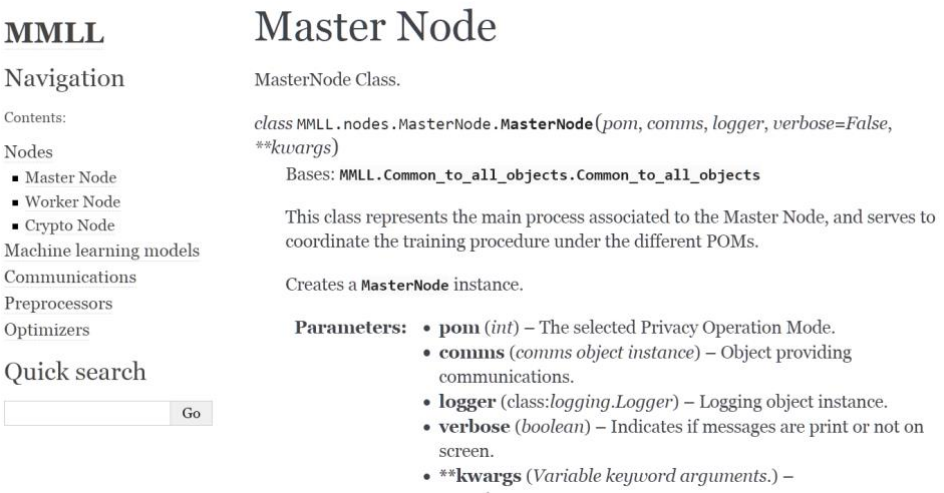

Figure 8 MMLL documentation: Master Node.

MMLL

Navigation

Contents:

Nodes
Machine learning models
Communications
Preprocessors
- Conversion to numeric
- Missing data imputation
- Deep learning
- Feature extraction
- Image reshape
- Image to vector
- Logarithmic scale
- Noise injection
- Normalization
- Outlier clipping
- Principal Component Analysis
- Random projection
- Sparse random projection
- Term Frequency - Inverse Document Frequency

Preprocessors

Contents:

- Conversion to numeric
- Missing data imputation
- Deep learning
- Feature extraction
- Image reshape
- Image to vector
- Logarithmic scale
- Noise injection
- Normalization
- Outlier clipping
- Principal Component Analysis
- Random projection
- Sparse random projection
- Term Frequency - Inverse Document Frequency

MMLL

Navigation

Contents:

Nodes
Machine learning models
Communications
Preprocessors
- Conversion to numeric
- Missing data imputation
- Deep learning
- Feature extraction
- Image reshape
- Image to vector
- Logarithmic scale
- Noise injection

Deep learning

Preprocessing object for deep learning image transformation @author: Angel Navia Vázquez

class
`MMLL.preprocessors.deep_learning.deep_learning_model`(*data_description*)

Bases: `object`

Parameters: **input_data_description** (*dict*) – Description of the input features

`transform`(*X*)

Transform data with a Deep Learning preprocessing

Parameters: **X** (*ndarray*) – Matrix with the input values
Returns: **transformed values**
Return type: ndarray

MMLL

Navigation

Contents:

Nodes
Machine learning models
Communications
Preprocessors
- Conversion to numeric
- Missing data imputation
- Deep learning
- Feature extraction
- Image reshape
- Image to vector
- Logarithmic scale
- Noise injection
- Normalization

Feature extraction

Preprocessing object for extracting given features @author: Angel Navia Vázquez

class
`MMLL.preprocessors.feature_extract.feature_extract_model`(*selected_features*, *input_data_description*)

Bases: `object`

Parameters:
- **selected_features** (*list of indices*) – Features to be retained
- **input_data_description** (*dict*) – Description of the input features

`transform`(*X*)

Transform data by extracting features

Parameters: **X** (*ndarray*) – Matrix with the input values
Returns: **transformed values**
Return type: ndarray

Figure 9 MMLL documentation: available pre-processors and two detailed examples.

MMLL

Navigation

Contents:

Nodes
Machine learning models
- Privacy Operation Mode 1
- Privacy Operation Mode 2
- Privacy Operation Mode 3
- Privacy Operation Mode 4
- Privacy Operation Mode 5
- Privacy Operation Mode 6

Communications
Preprocessors
Optimizers

Quick search

Go

Machine learning models

Contents:

- Privacy Operation Mode 1
  - Common
  - Kmeans
  - Neural networks
  - Support Vector Machine
  - Federated Budget Support Vector Machine
  - Distributed Support Vector Machine
- Privacy Operation Mode 2
  - Common
  - Kmeans
  - Neural networks
  - Support Vector Machine
  - Federated Budget Support Vector Machine
- Privacy Operation Mode 3
  - Common
  - Kmeans
  - Neural networks
  - Support Vector Machine
  - Federated Budget Support Vector Machine
- Privacy Operation Mode 4
  - Common
  - Linear Regression
  - Logistic Classifier
  - Multiclass Logistic Classifier
  - Clustering Kmeans
  - Kernel Regression
  - Budget Distributed Support Vector Machine
- Privacy Operation Mode 5
  - Common
  - Multiclass Logistic Classifier
  - Logistic Classifier
  - Multiclass Logistic Classifier
  - Clustering Kmeans
  - Kernel Regression
  - Budget Distributed Support Vector Machine
  - Multiclass Budget Distributed Support Vector Machine
- Privacy Operation Mode 6
  - Common
  - Ridge Regression
  - Logistic Classifier
  - Multiclass Logistic Classifier
  - Clustering Kmeans
  - Kernel Regression
  - Budget Distributed Support Vector Machine
  - Multiclass Budget Distributed Support Vector Machine

Figure 10 MMLL documentation: available ML models at every POM.

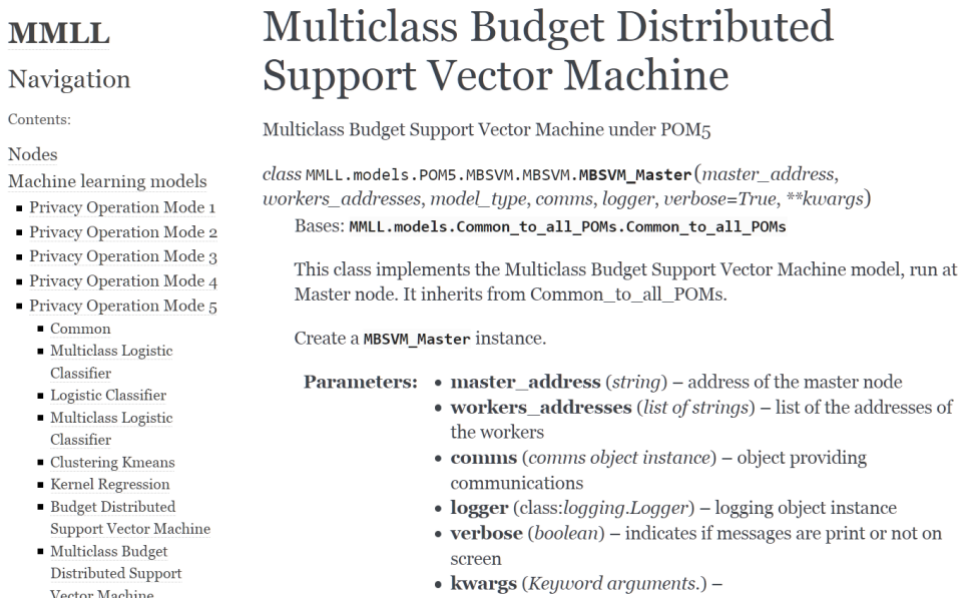

Figure 11 MMLL documentation: sample model documentation.

6.2 Demos execution

In order to be able to run the demos, the user has first to clone the repository for the demos. Working on the same virtual environment created in the above section, the user has to type:

```
git clone https://github.com/Musketeer-H2020/MMLL-demo.git
```

After that, navigate to the root directory of the repository and install the requirements:

```
cd MMLL-demo

pip install -r requirements.txt
```

The repository for the demonstration has two important folders:

- **demos**: This folder contains all the demos available for the machine learning library as well as the pre-processing algorithms described in D4.3. The different demos are organized by POMs and in order to be able to run them the pycloudmessenger json credentials file needs to be placed inside demos/demo_pycloudmessenger.
- **input_data**: Folder containing all the open datasets used for the different demos in pkl format. The datasets have to be downloaded from https://drive.google.com/drive/folders/1-piNDL_tL6V4pCI-En02zeCEqoL-dUUu?usp=sharing and placed in the input_data/ folder.

At this point, the virtual environment is ready and the user can execute the demos. Inside the folder for each demo the user can find additional instructions for execution in a README.txt.

Additionally, after launching the scripts a new folder, /results, is created. It contains the following subfolders:

- **figures**: Contains pictures used for the evaluation of the models.
- **logs**: Details with the logs of the execution.
- **models**: Stored models after the training is complete.

7 MMLL demonstration scripts

In this section we provide a categorized list of demos, organized by POM. In every one of the demo examples below, it is necessary to run every provided script line[2] in a separate console. The demo experiments are to be run using 5 data providers (5 worker nodes, every provided training dataset has been split into 5 separate participants). The provided demos are not prepared to operate on a different simulation conditions but the interested reader may easily provide its own dataset partition, use a different data Connector and run the MMLL experiments using an arbitrary number of participants.

7.1 POM4 demo scripts

7.1.1 Linear Regression (LR)

```
python3 pom4_LR_master_pycloudmessenger.py --dataset redwine --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom4_LR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 0

python3 pom4_LR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 1

python3 pom4_LR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 2

python3 pom4_LR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 3

python3 pom4_LR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 4

python3 pom4_LR_crypto_pycloudmessenger.py --verbose 1 --id 5
```

7.1.2 Kernel Regression (KR)

```
python3 pom4_KR_master_pycloudmessenger.py --dataset sinc1D --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom4_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id

python3 pom4_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 1
```

[2] All the code (demos and libraries) is python 3, we use "python3" in the scripts, but the Windows users may possibly need to use "python", even though we are always referring to execution under Python 3.

```
python3 pom4_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 2

python3 pom4_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 3

python3 pom4_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 4

python3 pom4_KR_crypto_pycloudmessenger.py --verbose 1 --id 5
```

7.1.3 Logistic Classifier (LC)

```
python3 pom4_LC_master_pycloudmessenger.py --dataset pima --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom4_LC_worker_pycloudmessenger.py --dataset pima --verbose 1 --id 0

python3 pom4_LC_worker_pycloudmessenger.py --dataset pima --verbose 1 --id 1

python3 pom4_LC_worker_pycloudmessenger.py --dataset pima --verbose 1 --id 2

python3 pom4_LC_worker_pycloudmessenger.py --dataset pima --verbose 1 --id 3

python3 pom4_LC_worker_pycloudmessenger.py --dataset pima --verbose 1 --id 4

python3 pom4_LC_crypto_pycloudmessenger.py --verbose 1 --id 5
```

7.1.4 Multiclass Logistic Classifier (MLC)

```
python3 pom4_MLC_master_pycloudmessenger.py --dataset M-iris --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom4_MLC_worker_pycloudmessenger.py --dataset M-iris --verbose 1 --id 0

python3 pom4_MLC_worker_pycloudmessenger.py --dataset M-iris --verbose 1 --id 1

python3 pom4_MLC_worker_pycloudmessenger.py --dataset M-iris --verbose 1 --id 2

python3 pom4_MLC_worker_pycloudmessenger.py --dataset M-iris --verbose 1 --id 3

python3 pom4_MLC_worker_pycloudmessenger.py --dataset M-iris --verbose 1 --id 4

python3 pom4_MLC_crypto_pycloudmessenger.py --verbose 1 --id 5
```

7.1.5 Clustering (K-means)

```
python3 pom4_Kmeans_master_pycloudmessenger.py --dataset synth2D --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom4_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 0
```

```
python3 pom4_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 1

python3 pom4_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 2

python3 pom4_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 3

python3 pom4_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 4

python3 pom4_Kmeans_crypto_pycloudmessenger.py --verbose 1 --id 5
```

7.1.6 Budget Support Vector Machine (BSVM)

```
python3 pom5_BSVM_master_pycloudmessenger.py --dataset synth2D-class --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom5_BSVM_worker_pycloudmessenger.py --id 0 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 1 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 2 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 3 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 4 --dataset synth2D-class --
verbose 1
```

7.2 POM5 demo scripts

7.2.1 Linear Regression (LR)

```
python3 pom5_LR_master_pycloudmessenger.py --dataset redwine --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom5_LR_worker_pycloudmessenger.py --id 0 --dataset redwine --verbose 1

python3 pom5_LR_worker_pycloudmessenger.py --id 1 --dataset redwine --verbose 1

python3 pom5_LR_worker_pycloudmessenger.py --id 2 --dataset redwine --verbose 1

python3 pom5_LR_worker_pycloudmessenger.py --id 3 --dataset redwine --verbose 1

python3 pom5_LR_worker_pycloudmessenger.py --id 4 --dataset redwine --verbose 1
```

7.2.2 Kernel Regression (KR)

```
python3 pom5_KR_master_pycloudmessenger.py --dataset sinc1D --verbose 1
```

```
(wait for the master to start listening for workers/cryptonode...)

python3 pom5_KR_worker_pycloudmessenger.py --id 0 --dataset sinc1D --verbose 1

python3 pom5_KR_worker_pycloudmessenger.py --id 1 --dataset sinc1D --verbose 1

python3 pom5_KR_worker_pycloudmessenger.py --id 2 --dataset sinc1D --verbose 1

python3 pom5_KR_worker_pycloudmessenger.py --id 3 --dataset sinc1D --verbose 1

python3 pom5_KR_worker_pycloudmessenger.py --id 4 --dataset sinc1D --verbose 1
```

7.2.3 Logistic Classifier (LC)

```
python3 pom5_LC_master_pycloudmessenger.py --dataset pima --verbose 1

(wait for the master to start listening for workers...)

python3 pom5_LC_worker_pycloudmessenger.py --id 0 --dataset pima --verbose 1

python3 pom5_LC_worker_pycloudmessenger.py --id 1 --dataset pima --verbose 1

python3 pom5_LC_worker_pycloudmessenger.py --id 2 --dataset pima --verbose 1

python3 pom5_LC_worker_pycloudmessenger.py --id 3 --dataset pima --verbose 1

python3 pom5_LC_worker_pycloudmessenger.py --id 4 --dataset pima --verbose 1
```

7.2.4 Multiclass Logistic Classifier (MLC)

```
python3 pom5_MLC_master_pycloudmessenger.py --dataset M-iris --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom5_MLC_worker_pycloudmessenger.py --id 0 --dataset M-iris --verbose 1

python3 pom5_MLC_worker_pycloudmessenger.py --id 1 --dataset M-iris --verbose 1

python3 pom5_MLC_worker_pycloudmessenger.py --id 2 --dataset M-iris --verbose 1

python3 pom5_MLC_worker_pycloudmessenger.py --id 3 --dataset M-iris --verbose 1

python3 pom5_MLC_worker_pycloudmessenger.py --id 4 --dataset M-iris --verbose 1
```

7.2.5 Clustering (K-means)

```
python3 pom5_Kmeans_master_pycloudmessenger.py --dataset synth2D --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom5_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 0
```

```
python3 pom5_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 1

python3 pom5_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 2

python3 pom5_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 3

python3 pom5_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 4
```

7.2.6 Budget Support Vector Machine (BSVM)

```
 python3 pom5_BSVM_master_pycloudmessenger.py --dataset synth2D-class --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom5_BSVM_worker_pycloudmessenger.py --id 0 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 1 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 2 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 3 --dataset synth2D-class --
verbose 1

python3 pom5_BSVM_worker_pycloudmessenger.py --id 4 --dataset synth2D-class --
verbose 1
```

7.2.7 Multiclass Budget Support Vector Machine (MBSVM)

```
python3 pom5_MBSVM_master_pycloudmessenger.py --dataset M-iris --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom5_MBSVM_worker_pycloudmessenger.py --id 0 --dataset M-iris --verbose 1

python3 pom5_MBSVM_worker_pycloudmessenger.py --id 1 --dataset M-iris --verbose 1

python3 pom5_MBSVM_worker_pycloudmessenger.py --id 2 --dataset M-iris --verbose 1

python3 pom5_MBSVM_worker_pycloudmessenger.py --id 3 --dataset M-iris --verbose 1

python3 pom5_MBSVM_worker_pycloudmessenger.py --id 4 --dataset M-iris --verbose 1
```

7.3 POM6 demo scripts

7.3.1 Ridge Regression (RR)

```
python3 pom6_RR_master_pycloudmessenger.py --dataset redwine --verbose 1
```

```
(wait for the master to start listening for workers/cryptonode...)

python3 pom6_RR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 0

python3 pom6_RR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 1

python3 pom6_RR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 2

python3 pom6_RR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 3

python3 pom6_RR_worker_pycloudmessenger.py --dataset redwine --verbose 1 --id 4
```

7.3.2 Kernel Regression (KR)

```
python3 pom6_KR_master_pycloudmessenger.py --dataset sinc1D --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom6_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 0

python3 pom6_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 1

python3 pom6_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 2

python3 pom6_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 3

python3 pom6_KR_worker_pycloudmessenger.py --dataset sinc1D --verbose 1 --id 4
```

7.3.3 Logistic Classifier

```
python3 pom6_LC_master_pycloudmessenger.py --dataset pima --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom6_LC_worker_pycloudmessenger.py --id 0 --dataset pima --verbose 1

python3 pom6_LC_worker_pycloudmessenger.py --id 1 --dataset pima --verbose 1

python3 pom6_LC_worker_pycloudmessenger.py --id 2 --dataset pima --verbose 1

python3 pom6_LC_worker_pycloudmessenger.py --id 3 --dataset pima --verbose 1

python3 pom6_LC_worker_pycloudmessenger.py --id 4 --dataset pima --verbose 1
```

7.3.4 Multiclass Logistic Classifier (MLC)

```
python3 pom6_MLC_master_pycloudmessenger.py --dataset M-iris --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom6_MLC_worker_pycloudmessenger.py --id 0 --dataset M-iris --verbose 1

python3 pom6_MLC_worker_pycloudmessenger.py --id 1 --dataset M-iris --verbose 1
```

```
python3 pom6_MLC_worker_pycloudmessenger.py --id 2 --dataset M-iris --verbose 1

python3 pom6_MLC_worker_pycloudmessenger.py --id 3 --dataset M-iris --verbose 1

python3 pom6_MLC_worker_pycloudmessenger.py --id 4 --dataset M-iris --verbose 1
```

7.3.5 Clustering (K-means)

```
python3 pom6_Kmeans_master_pycloudmessenger.py --dataset synth2D --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom6_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 0

python3 pom6_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 1

python3 pom6_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 2

python3 pom6_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 3

python3 pom6_Kmeans_worker_pycloudmessenger.py --dataset synth2D --verbose 1 --id 4
```

7.3.6 Budget Support Vector Machine (BSVM)

```
python3 pom6_BSVM_master_pycloudmessenger.py --dataset synth2D-class --verbose 1

(wait for the master to start listening for workers/cryptonode...)

python3 pom6_BSVM_worker_pycloudmessenger.py --dataset synth2D-class --verbose 1 --
id 0

python3 pom6_BSVM_worker_pycloudmessenger.py --dataset synth2D-class --verbose 1 --
id 1

python3 pom6_BSVM_worker_pycloudmessenger.py --dataset synth2D-class --verbose 1 --
id 2

python3 pom6_BSVM_worker_pycloudmessenger.py --dataset synth2D-class --verbose 1 --
id 3

python3 pom6_BSVM_worker_pycloudmessenger.py --dataset synth2D-class --verbose 1 --
id 4
```

7.3.7 Multiclass Budget Support Vector Machine (MBSVM)

```
python3 pom6_MBSVM_master_pycloudmessenger.py --dataset M-iris --verbose 1

(wait for the master to start listening for workers/cryptonode...)
```

```
python3 pom6_MBSVM_worker_pycloudmessenger.py --id 0 --dataset M-iris --verbose 1

python3 pom6_MBSVM_worker_pycloudmessenger.py --id 1 --dataset M-iris --verbose 1

python3 pom6_MBSVM_worker_pycloudmessenger.py --id 2 --dataset M-iris --verbose 1

python3 pom6_MBSVM_worker_pycloudmessenger.py --id 3 --dataset M-iris --verbose 1

python3 pom6_MBSVM_worker_pycloudmessenger.py --id 4 --dataset M-iris --verbose 1
```

8 Conclusions

In this deliverable (D4.7) we have presented the final version of the MUSKETEER Machine Learning Library under POMs 4, 5 and 6 (MMLL V2.0). We have implemented Linear/Logistic models, Clustering (Kmeans) and Kernel/Support Vector Machines methods. This version of the library is fully compliant with the current "pycloudmessenger" service (IBM Cloud communication service) and it has also been successfully integrated in the MUSKETEER Client Connector developed in WP7. The performance and characteristics of the implemented algorithms will be evaluated in the context of WP6.