Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)
Grant No 824988

D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial version

October 19 July 19
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version
Executive Summary

This deliverable (D4.2 "Pre-processing, normalization, data alignment and data value estimation") is the first software outcome of MUSKETEER’s WP4 in the form of a Demonstrator. Besides the software prototype itself, it comprises a report with the description of the operation of the MUSKETEER Demonstrator under different operations, as well as the software documentation and description of the software components. The implementations included in the Demonstrator are preliminary yet fully operative, although possibly their design may change as the project progresses. In future versions, more operations and modules will be available, some of the existing modules could be replaced by improved versions (the currently available algorithms are rather simple), and some general redesign may be necessary to facilitate the integration with the rest of MUSKETEER components.

Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Author</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01 July 2019</td>
<td>For internal review</td>
<td>Angel Navia-Vázquez</td>
<td>First draft</td>
</tr>
<tr>
<td>2</td>
<td>10 July 2019</td>
<td>Review inputs</td>
<td>Chiara Napione</td>
<td>Update</td>
</tr>
<tr>
<td>3</td>
<td>13 July 2019</td>
<td>Review inputs</td>
<td>Mathieu Sinn</td>
<td>Update</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Final Version</td>
<td></td>
<td>Update</td>
</tr>
<tr>
<td>5</td>
<td>15 July 2019</td>
<td>Clean and submis-</td>
<td>Gal Weiss</td>
<td>Final</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sion</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

**LIST OF FIGURES** ........................................................................................................... 5

**LIST OF ACRONYMS AND ABBREVIATIONS** ................................................................. 7

1 **INTRODUCTION** ........................................................................................................... 8
   1.1 Purpose ...................................................................................................................... 8
   1.2 Related Documents .................................................................................................... 8
   1.3 Document Structure ................................................................................................. 9

2 **CONTEXT OF THE DEMONSTRATOR** ..................................................................... 10
   2.1 Task Alignment approaches .................................................................................... 12
   2.2 Data value estimation approaches .......................................................................... 14

3 **DEMONSTRATOR ASSUMPTIONS** .......................................................................... 18

4 **INSTALLATION INSTRUCTIONS** .............................................................................. 22

5 **EXECUTION SETUP** .................................................................................................. 23

6 **OPERATION OF THE DEMONSTRATOR** ................................................................. 25
   6.1 User interface ........................................................................................................... 25
   6.2 Master node initialization ......................................................................................... 26
   6.3 Display available end users .................................................................................... 27
   6.4 Task alignment ......................................................................................................... 27
   6.5 Excluding unwanted users ...................................................................................... 28
   6.6 Ad Hoc local data pre-processing ........................................................................... 29
   6.7 Global data normalization ......................................................................................... 32
   6.8 Global feature selection ............................................................................................ 33
   6.9 Data value estimation ............................................................................................... 34
   6.10 Model training ........................................................................................................ 35
6.11 Performance on local validation/test ................................................. 36
6.12 Performance on training data at every user ........................................ 36
6.13 Create ROC figures ........................................................................... 37
6.14 Terminate all user nodes ...................................................................... 39
6.15 Exit MUSKETEER Demonstrator ......................................................... 39

7 SOFTWARE DOCUMENTATION .................................................................. 40

8 CONCLUSIONS ......................................................................................... 54

9 REFERENCES ............................................................................................... 54
List of Figures

Figure 1 MUSKETEER’s PERT diagram ................................................................. 8
Figure 2 The centralized (a) vs. (privacy/confidentiality preserving) distributed scenario (b). Every user provides a portion of the training dataset .................................................. 10
Figure 3 The Task alignment and data value estimation problems .............................. 11
Figure 4 Task alignment scenarios: supervised (a) vs. unsupervised (b) ......................... 13
Figure 5 Task alignment estimation in the supervised scenario with the correlation (a) and gradients (b) .................................................................................................................. 14
Figure 6 Data value estimation using the full model (a) and the fast linear estimation approach (b) .................................................................................................................. 16
Figure 7 Greedy incremental Data value estimation using the Final Model Inter-Dependent User approach .................................................................................................................. 17
Figure 8 Data value estimation based on fast correlation and gradient measurements ...... 18
Figure 9 The Demonstrator setup: Master, Workers and other participating objects .......... 20
Figure 10 The States and transitions of the Master FSM showing operations for task alignment, data value estimation and pre-processing ..................................................... 21
Figure 11 The States and transitions of the Master FSM showing operations for model training and computation of the performance at workers ................................................. 21
Figure 12 The States and transitions of the Worker FSM showing operations for task alignment, data value estimation and pre-processing ..................................................... 22
Figure 13 The States and transitions of the Workers FSM showing operations for model training and computation of the performance at workers ................................................. 22
Figure 14 The Demonstrator User Interface .................................................................. 24
Figure 14 The terminal showing messages from user "0" .............................................. 25
Figure 16 The Master Node is initiated ........................................................................ 26
Figure 17 listing the connected users ......................................................................... 26
Figure 18 Applying the task alignment estimation ........................................................ 27
Figure 19 Excluding misaligned users ........................................................................... 28
Figure 20 Misaligned users are disconnected ............................................................... 29
Figure 21 Updated list of participants ......................................................................... 29
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version

Figure 22 Local Pre-processing Menu ................................................................. 30
Figure 23 AlexNet network used for Deep Learning pre-processing ................... 30
Figure 24 Applying the down-sample pre-processing ........................................... 31
Figure 25 Workers applying the pre-processing and showing the new data size ....... 31
Figure 26 Applying the Deep Learning Pre-processing ......................................... 32
Figure 27 Global normalization parameters estimation and application to users’ data ..... 33
Figure 28 Applying the global greedy feature selection ....................................... 34
Figure 29 Data Value estimation ........................................................................... 35
Figure 30 Model training ....................................................................................... 35
Figure 31 Performance on local data ..................................................................... 36
Figure 32 Performance on users’ data ................................................................. 37
Figure 33 Comparative ROC curves for different solutions on the validation set ...... 38
Figure 34 Comparative ROC curves for different solutions on the test set .......... 39
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC</td>
<td>Area Under (ROC) Curve</td>
</tr>
<tr>
<td>CA</td>
<td>Consortium Agreement</td>
</tr>
<tr>
<td>DP</td>
<td>Differential Privacy</td>
</tr>
<tr>
<td>DC</td>
<td>Data Connector</td>
</tr>
<tr>
<td>DV</td>
<td>Data Value</td>
</tr>
<tr>
<td>FLE</td>
<td>Fast Linear Estimation</td>
</tr>
<tr>
<td>FS</td>
<td>Feature Selection</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>GA</td>
<td>Grant Agreement</td>
</tr>
<tr>
<td>IDR</td>
<td>Intermediate Data Representation</td>
</tr>
<tr>
<td>LC</td>
<td>Logistic Classifier</td>
</tr>
<tr>
<td>LGFS</td>
<td>Linear Greedy Feature Selection</td>
</tr>
<tr>
<td>MK</td>
<td>Master Key</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layer Perceptron</td>
</tr>
<tr>
<td>MN</td>
<td>Master Node</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PERT</td>
<td>Program evaluation and review technique</td>
</tr>
<tr>
<td>PK</td>
<td>Public Key</td>
</tr>
<tr>
<td>POM</td>
<td>Privacy Operation Mode</td>
</tr>
<tr>
<td>PP</td>
<td>Privacy Preserving</td>
</tr>
<tr>
<td>PPML</td>
<td>Privacy Preserving Machine Learning (a.k.a. Privacy Preserving Data Mining)</td>
</tr>
<tr>
<td>ROC</td>
<td>Receiver Operating Characteristics</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TA</td>
<td>Task Alignment</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>WN</td>
<td>Worker Node</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Purpose

This deliverable provides a Demonstrator of some of the MUSKETEER capabilities (mainly focusing on data pre-processing, data normalization, data alignment and data value estimation). Although not required in the Agreement, we also provide a very simple Machine Learning (ML) model operating under confidentiality preserving conditions, such that the effects of the pre-processing operations can be measured and quantified in a particular task (image classification) using Receiver Operating Characteristics (ROC) curves on both validation and test sets.

1.2 Related Documents

D4.2 will serve as a basis for a deeper understanding of the models and algorithms to be further developed in WP4 (D4.4 and D4.6), and it will provide valuable information to WP7 (Client side connector implementation and use case piloting), as indicated in the PERT diagram below. In general terms, it constitutes a tangible outcome such that any member of the consortium can carry out "hands on" experimentation with a preliminary version of the platform, and such interaction could provide relevant feedback among all participants.

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 "Context of the Demonstrator" briefly revisits the main objectives of MUSKETEER from a Machine Learning point of view. We focus on one possible particular scenario (user story) and the main challenges that have been identified. We briefly describe preliminary approaches to deal with the task alignment and data value estimation problems, although the main research about these topics will take place in the upcoming months, until the final versions are delivered in month 26 (D4.3).
- The section "Demonstrator assumptions" describes the specific parameters used in the demonstrator: models, task and dataset description, number of users, available operations, etc.
- In Section 4: "Execution Setup", we explain the basic steps for executing the demonstrator in a multicore machine. Currently, it is only possible to run the demonstrator in a single machine, but when a new communications library is available, the same software demonstrator could be run in different machines, simply replacing the communications module with the new one.
- The Section "Operation of the Demonstrator" describes a text-based basic User Interface (only for the purpose of this demonstrator) and the available options in the "Menu". We illustrate the effect of every option in every participant, and show the results. This section could serve as a script to execute a demonstration with the provided software prototype.
- Finally, although the full software documentation is included in the software package in html format (and it will be dynamically updated), we have also included in this report part of that documentation, to facilitate a preliminary inspection of the software components.

1 The communications library will be developed under WP3 and it will ultimately support communication between processes in different machines, hosted possibly in different organizations or even countries, via a cloud-based message broker.
2 Context of the demonstrator

The Demonstrator described in this document is a simplified instance of the MUSKETEER platform, mainly concentrating on illustrating the concepts of pre-processing, normalization, task alignment and data value estimation. The MUSKETEER platform aims at solving Machine Learning (ML) problems using data from different users while preserving the privacy/confidentiality of the data. Essentially, it aims at deploying a distributed ML setup (Figure 1(b)) such that a model equivalent to the one obtained in the centralized setup (Figure 1(a)) is obtained.

The centralized solution requires that the data from different users is 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 exchange some information (intermediate data representation, IDR) among the participating users such

![Figure 2](image.png)

Figure 2 The centralized (a) vs. (privacy/confidentiality preserving) distributed scenario (b). Every user provides a portion of the training dataset.

It would be possible to gather data from different users in the same place if it is previously "transformed" under the "data outsourcing" paradigm, to be finally processed in a central location, a cloud system, for instance. If the transformation follows Differential Privacy principles, the data is altered and the effect on the final models is uncertain. If the data is encrypted, there is a computational overhead during the training phase, as well as complications when processing data encrypted with different keys. MUSKETEER does not currently contemplate the "transformed data outsourcing" scheme.

Any intermediate data representation should carry some information about the data it is derived from (to allow learning), while hiding the actual raw data values to the participants in the protocol. Averaged gradients, auto-correlation matrices or cross-correlation vectors could be examples of IDR, each one revealing different partial information about the datasets.
that a Master Node (MN) obtains the final ML model without ever receiving/seeing the raw data of the users.

Under this setup, several users may want to cooperate to train a joint model, but it the amount and quality of the data provided by every one of them is not known beforehand. Actually, some of them could even be malicious, and try to inject low quality or directly "poisonous" data into the learning process. Obviously, all of them could declare that they are contributing with a large amount of high quality data, but that is something to be further assessed by the Master node controlling the learning process, since in a data economy, an economic reward must be sent to every participant, according to their real contribution (Data Value). Therefore, the tasks of detecting the ‘alignment of every participant’ with respect to the defined task is a very important preliminary step, since the inclusion of "bad" data in the learning process could largely bias the resulting model. A thorough analysis of these types of attacks and of potential strategies for defending against them will be carried out in the context of WP5: Security and Trustworthiness of Federated Machine Learning Algorithms, and the results produced in WP5 will help reducing the risk of incorporating data that may damage the learning process.

By now, and for illustration purposes, we will present very basic Task Alignment (TA) approaches. For instance, in Figure 3 below, user #3 offers a large amount of data, but its quality is low and therefore the reward should be smaller than for users #1 and #2 (better data quality), even considering the latter contribute with less data. User #4 offers data, but it is detected as non-aligned with the task to be solved, and therefore it is considered as rubbish or poisoned data. User #4 should be excluded from the learning process as soon as possible, and obviously gets no reward in return.

![Figure 3 The Task alignment and data value estimation problems](image-url)
2.1 Task Alignment approaches

The first step before training the model is to evaluate if all the contributed data corresponds to the same task. The participating users may provide data with different quality, and some of them could even be malicious and provide wrong data. This step is described as "task alignment", i.e., we want to estimate if every users data is "in line" with the real task to be solved.

In a traditional setup, where all data is available for analysis without any kind of restriction, the task alignment problems could be stated as the problem of verifying that the different datasets share a common (or at least similar) joint distribution. Actually, the main objective at this stage is to detect distributions that significantly differ from the reference one. This is, if data from user "0" can be described as \( \{x_p, y_p\}^0_0, p = 1, ..., N_0 \), we can assume that the patterns associated to user "0" come from a given joint distribution \( p_0(x, y) \). We could apply the same reasoning to the data coming from the other users \( p_1(x, y) \), \( p_2(x, y) \), etc., including the one providing the reference: \( p_R(x, y) \). Therefore, we could state the task alignment problem as a distance computation between those distributions: the users with distributions closer to the reference one are considered "aligned", and we assume that they will contribute positively to the training process. Misaligned users should be removed from the training set. Classical examples of such a kind of distances are, among many other:

- Kullback–Leibler divergence [Kullback_1951]
- Hellinger distance [Hellinger_1909]
- Total variation distance [Chatterjee_2008]
- Jensen–Shannon divergence [Schütze_1999]
- Wasserstein metric (earth mover’s distance) [Rüschendorf_2001]

However, in the MUSKETEER context, we do not have an unlimited access to the data, and estimations of the distributions are not available. We will have to restrict ourselves to computing distances among the intermediate data representations (IDR) that preserve the privacy/confidentiality of the data.

We will assume therefore that some representation vector can be constructed from the available information from every user (correlation matrices/vectors, gradient aggregation, data distribution, etc.), and such IDR can be shared with the Master Node without compromising the privacy/confidentiality. We will propose here some preliminary IDR, but a deeper research will be carried out during the next months in relation to the concept of “sufficient statistics” and to what degree those IDR may reveal unwanted information about the users data.
Under this hypothesis, the alignment can be defined as any measurement among those IDR vectors, for instance, normalized matrix distances, cosine of the angle among those vectors, etc., or any other statistic measure able to operate under the privacy/confidentiality requirements. This is something to be investigated in the upcoming months, and it is an open research issue how the distances among the original distributions are preserved in the transformed (IDR) space. As an example, we show here two possible IDR vector computations:

a) **IDR based on correlation matrices/vectors.** We assume that the master node receives a correlation matrix and cross-correlation vector from every one of the workers \((R_1, r_1), \ldots, (R_m, r_m)\). A possible IDR is to compute the following vector:

\[
I_m = \begin{bmatrix} R_m \\ r_m \end{bmatrix}, \text{ where } [.] \text{ represents the operation of vertically stacking the columns of a matrix or several vectors to build a larger vector.}
\]

b) **IDR based on gradients.** We assume that the master node receives an accumulated gradient vector from every one of the workers \((R_1, r_1), \ldots, (R_m, r_m)\) to update the model. A possible option here is to directly use the accumulated gradient vector as IDR.

Based on the above mentioned IDR vectors, two scenarios are possible. In the first one (supervised) we assume that a reference is available to estimate the correct alignment. This is possible when the task proponent also provides a validation data set such that the reference IDR vector is the one derived from that validation dataset, and it can be used as a reference or target IDR to be followed by the other participants, as shown in Figure 4 (a)

![Figure 4 Task alignment scenarios: supervised (a) vs. unsupervised (b)](image)

In this supervised scenario (Figure 4a), in spite of the presence of many other mis-aligned users, the aligned ones are always correctly identified, and the final estimated alignment is very close to the correct one. The second scenario (Figure 4b), named as unsupervised, does not provide a ground truth reference, and we need to trust on a majority vote approach. In
the case of many colluding users towards a wrong solution, the system is not able to detect the correct alignment. If the number of "fair" users is larger than the wrong ones, the solution will also be correct, as in the supervised case\textsuperscript{4}.

As an example, in a supervised task alignment scenario (where a reference is available), and 6 users (3 of them with fair data), a possible task alignment estimation results could be as follows:

We can observe how users No. 1, 4 and 5 are not aligned (their agreement/alignment with the task is lower). These users should be removed from the platform, to protect the training process.

After this preliminary detection, any procedure provided in WP5 (Security and Trustworthiness) could improve the detection of malicious users trying to interfere in the normal training process in more subtle ways (data poisoning, evasion attacks, etc.).

\subsection{Data value estimation approaches}

The Data Value (DV) estimation problem is related to the task alignment one. After excluding the misaligned users (users 1, 4, 5 in the example of the previous section, continued here), the objective is to assign a percentage of merit to every "fair" contributing user (users No. 0, 2, 3).

We foresee the following families of approaches for DV estimation, their detailed formulation, performance and other characteristics -such as computational complexity-, are still to be investigated.

We can classify the approaches according to the relationship among users data during the DV estimation:

\textsuperscript{4} The threat by colluding malicious users will be extensively analysed in WP5.
• **User independent methods**: the DV of every user is estimated uniquely using data from that user, independently from the rest of the users (absolute measures)

• **User inter-dependent methods**: the DV contribution of every user depends on the contributions of the other users (relative measures)

With respect to the models used during the process:

• **Final/Full Model approach (FM)**: the ML model architecture to be finally obtained is used in every intermediate step of the DV estimation process, i.e., if a Multi-Layer Perceptron (MLP) is the model to be ultimately obtained, then, any intermediate model used will also be an MLP.

• **Fast Linear Estimation (FLE)**: a linear model is used for the intermediate estimations, even when the final one is a different one. Faster but less accurate estimations are expected to be obtained.

In what follows, we will briefly describe some of the approach combinations and the obtained results.

Under a **Full Model User Independent** approach we directly use the performance achieved with the targeted ML model (in the case illustrated here, a Logistic Classifier, LC) to estimate the real contribution of every participant to the final solution. The performance obtained with the LC is shown in Figure 6(a), where the Area Under Curve (AUC) values are shown for every “fair” user (users 0, 2 and 3). These values indicate which user is providing the most valuable dataset (users 0 and 3 are equally best, user 2 is slightly worse).

The **Fast Linear Estimation User Independent** approach uses a –Least Squares- linear model, which can be trained much faster. The resulting AUC values are shown in Figure 6(b) and show a similar pattern.
The Data Value (in percentage) has been directly estimated here as the corresponding proportions for every user, i.e, for user "i" (i = 0, 2, 3):

$$DV_i = \frac{\text{AUC}_i}{\text{AUC}_0 + \text{AUC}_2 + \text{AUC}_3} \times 100$$

Using this approach we obtain the following DV estimations:

- **FM**: User "0": 33.4%, User "2": 33.2%, User "3": 33.4%
- **FLE**: User "0": 33.9%, User "2": 32.3%, User "3": 33.8%

These estimations are coherent with the “ground truth” of the experiment (these three users provide fair data, the user number 2 provides slightly less training patterns).

We observe that good estimates can be achieved using the FLE approach in comparison with the FM one. Anyhow, it is too early to draw any conclusion, and more research and experimentation is needed during the upcoming months before producing the final algorithms.

The above described methods only provide a DV estimation proportional to measurements on independently trained models, and they do not take into account the potential correlations among the datasets of the participants. An alternative is to use a **Full Model User Inter-Dependent** (brute force) approach, but it requires training a big number of final models with different input data configurations, with the corresponding overhead in computation. Under a greedy setup that avoids evaluating some of the possible combinations, the procedure would consist on the following steps:

1. Identify the user that provides the best data (user 3, in our case)
2. Evaluate different incremental models, by combining contributions from user 3 with the other. In our case we have these two options:
   a. Model “A” trained with data from users 3 and 0
   b. Model “B” trained with data from users 3 and 2
3. Choose the best performing model, in this case, model “A” is the best performing, as shown in Figure 7 below, and therefore the second best user is user No. “0”.
4. Repeat the process until all users have been evaluated and ranked.
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version

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

Figure 7 Greedy incremental Data value estimation using the Final Model Inter-Dependent User approach.

In this case, we assume that the master node (the task proponent) already has a trained model with AUC=0.94 that he/she wants to improve. The performance curve shown in Figure 7 represents the improved AUC values when the base model is retrained adding data from every user. We have sorted the users according to the largest improvement with respect to the reference solution. In Figure 7 we observe how data from user "3" provides an improvement up to AUC=0.95 from a previous AUC of the reference dataset of 0.94. The next users’ data helps to improve the solution up to a value of AUC = 0.956, and finally, the third user provides a marginal improvement. This "winner-takes-all" approach would yield a data value estimation very favorable to the first selected user:

- User "3": 68.7%, User "0": 24.3%, User "2": 7.0%

We still need to further investigate alternative schemes for data value assignation and their implications in a true data value market.

Finally, there is a second family of possible approaches for data value estimation, where it is not necessary to train full models, but rely on the alignment concept described in the previous section, directly measured on IDR values. This way, the users more "aligned" to the reference value (if available), can be assigned a larger data value. In Figure 8 below we show such estimation using two types of IDR: correlation matrices and aggregated gradients (remember that users 1, 4 and 5 have been removed due to their poor alignment).
Using these measurements, we obtain DV estimates very close to the ones obtained using the full models, under the independent scheme:

- **Corr.**: User "0": 36.3%, User "2": 32.1%, User "3": 31.6%
- **Grad.**: User "0": 30.4%, User "2": 26.8%, User "3": 42.8%

In the demonstrator we have included a preliminary fast estimation based on correlation measurements. In the final version of the platform, more estimation methods will be available. All the ML algorithms and estimation methods will be packaged in a library and as such they will become available for the end users of the MUSKETEER platform.

## 3 Demonstrator assumptions

In what follows, we assume that a Machine Learning task has already been defined, and that the platform has already identified all the potential users participating in the training process. In the complete, end-to-end version of the MUSKETEER platform, the services which allow users to register to the platform, define tasks and join tasks will be developed under WP3.

Therefore, for the purpose of this demonstrator, we will assume the following:

- **General description of the task**: an image classification problem is to be solved. The input images are handwritten digits of size 28x28 pixels and the task is to differentiate between images containing even and odd numbers. 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.). The source dataset used to build this example is the MNIST handwritten digits dataset.

- **User addresses and execution**: the list of addresses of the participating nodes (Master Node (MN) and Worker Nodes (WN)) is available. For the purpose of this demo, the address of the master is "5", and the end users providing training data have addresses "0", "1", "2", "3", "4". Every participant (Master/Workers) will be a separate process in a local machine. The current version of the Communications Library (CL) is primarily designed to communicate between processes in the same machine, and we have executed these simulations using processes in a single machine, but in the future the experiments will also cover different remote machines.

- **Data**: the data for training, validating and testing will be provided to MUSKETEER by means of a Data Connector (DC) specific for this Demonstrator. The DC in the demonstrator simply loads data from a file, but in the future any other compatible data connector can be used (SQL access, for instance). The input patterns (images reshaped to a 1-D vector) have a dimension of 784, and targets are binary. The features are the pixel values of the input images, but some transformation can be applied within the context of the demonstrator. For the sake of illustration, the TA and DV functionalities have been implemented in the demonstrator. We have designed a dataset partition as follows:
  - User "0" provides 4078 fair/correct/useful training patterns
  - User "1" provides 3340 random (both features and targets) training patterns
  - User "2" provides 5579 fair/correct/useful training patterns
  - User "3" provides 7773 fair/correct/useful training patterns
  - User "4" provides 4230 training patterns with correct features but with opposite targets

Therefore, users "1" and "4" do not contribute positively to the training task (something to be automatically detected during the task alignment phase), as it will be shown later. Users "0", "2" and "3" provide valuable data, and the value of their contribution is to be estimated during the Data Value estimation phase.

- **Confidentiality requirements**: In the Demonstrator we will assume that the raw data is never sent outside of the owner’s context and that the trained model is kept secret (only known to the task proposer). We will allow to exchange among the participants

---

some IDR, transformations of the data (such as aggregations, cross-correlation matrices, 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.

In the next Figure we show the main components (objects) in the Demonstrator and their inter-relationship during the normal operation. The Master node is controlling all the activity, mainly reacting to instructions received from the task proponent through the User Interface. The Workers mainly operate in a responsive way: when they receive a message or instruction, they perform a computation, return the result, and come back to a "listening state". This design is specific for the demonstrator, and the final design in MUSKETEER may be slightly different. However, we will use this preliminary approach as a baseline for discussion about different alternatives or designs.

![Diagram](image)

**Figure 9 The Demonstrator setup: Master, Workers and other participating objects**

The behaviour of the Master and Workers may become quite complex, depending on the selected POM and Machine Learning procedure to be implemented. To define and control such behaviour, we have implemented a Finite State Machine (FSM) object, running in both Master and Workers, such that the state of those machines determines which are the actions to be executed, and the received inputs (messages, instructions, conditions), indicates
when to change from one state to another. For illustration purpose, we include here some of the states and transitions in the Master. It is out of the scope of this document to enter into technical details about the explanation/interpretation of every transition/state. But, as an example, if the user wants to carry out a local preprocessing, the event “go_sending_prep_object” is activated in the Master, and the state is changed to “sending_prep_object” in Figure 10. As a consequence, every Worker receives a command that activates the event “go_storing_prep_object” in Figure 12, and the protocols continue according to the structure defined in the FSM. The complexity of these protocols will be hidden to the end user, only the ML designer will get access to this low level implementation detail.

Figure 10 The States and transitions of the Master FSM showing operations for task alignment, data value estimation and pre-processing

Figure 11 The States and transitions of the Master FSM showing operations for model training and computation of the performance at workers
Figure 12 The States and transitions of the Worker FSM showing operations for task alignment, data value estimation and pre-processing

Figure 13 The States and transitions of the Workers FSM showing operations for model training and computation of the performance at workers

4 Installation instructions

Before executing the Demonstrator, it is necessary to correctly configure an execution Python 3 environment with all the required libraries. In the final version of the platform, such configuration will be simplified, since the code will be embedded in a “docker” container.

Fully detailed installation instructions are included in the Software Documentations, but we describe here the general guidelines.

For the purpose of executing this demonstrator it is highly advisable to use the Anaconda python 3 distribution\(^6\), available for several Operating Systems (OS).

\(^6\) [https://www.anaconda.com/distribution/](https://www.anaconda.com/distribution/)
Once Anaconda is correctly installed, we need to open one “Anaconda Prompt Terminal” and execute a preliminary libraries update:

`conda update conda`

`conda update anaconda`

The safest way to execute the code is to define a specific execution environment, with all the required libraries. This procedure is slightly different, depending on the OS:

**Linux and macOS:**

`bash make_conda_environment_unix.sh`

Once the environment is ready, we activate it (we have to do this activation in every new terminal we open):

`conda activate Musk_Demo`

**Windows:**

`.\make_conda_environment_windows.bat`

## 5 Execution setup

We describe here the needed steps to execute the Demonstrator. These instructions are also included with greater detail in the software documentation (README file).

To execute the demonstrator and conveniently observe all the output messages, we will need to open 7 terminals on the same machine:

- **On terminal 1,** we execute "`python3 musketeer.py`", this process provides the basic communication facilities among the other processes.

- **On terminal 2,** we execute "`python3 master.py`", this window will show an elementary user interface to interact with the demonstrator, as shown below. This process will be controlling the behaviour of the other processes, as a response to the options introduced in the Menu.

---

7 Note that, depending on the OS, the Python executable could be named “python” or “python3”. In any case, the code is Python 3.
On terminals 3-7, we execute the end-user parts (5 different users):

"python3 worker.py --my_id 0 --model_type RidgeRegression"
"python3 worker.py --my_id 1 --model_type RidgeRegression"
"python3 worker.py --my_id 2 --model_type RidgeRegression"
"python3 worker.py --my_id 3 --model_type RidgeRegression"
"python3 worker.py --my_id 4 --model_type RidgeRegression"

Alternatively to this last step, a script that executes the 5 processes in the same terminal can be executed instead, and the behaviour of the Demonstrator would be the same, although the messages from all the users will be shown in the same window, which could be slightly confusing. The advantage of using this script is that we only need 3 terminals to run the demonstrator.

**In Linux/macOS:**

bash ./launch_workers_unix.sh

**In Windows:**

`\launch_workers_windows.bat`
If we opt for the 7-terminals approach, for instance, in the terminal for user "0", we observe how it starts all the needed elements, loads the data and enters the "listening state":

![Figure 1514 The terminal showing messages from user "0".](image)

By default, many messages are printed in the terminals during the operation of the Demonstrator. Those messages could be easily deactivated by setting “verbose = False”, but they are useful to understand the steps executed by every process in the Demonstrator. Anyhow, a log folder (placed in the Demonstrator folder) containing the resulting log files is also available. These log files show the messages produced by the communication library, the FSM and the ML code itself. These logs are produced even when messages on screen are deactivated.

At this point, the demonstrator is running and ready to operate on the users’ data. In the next section we will guide the reader through the currently supported actions.

## 6 Operation of the demonstrator

In what follows we will illustrate the operation of the demonstrator.

### 6.1 User interface

As described in the previous section, we provide a simple User Interface (UI) to facilitate the interaction with the Demonstrator and the evaluation of all its functionalities. The options offered in the "Menu" are:

- **Load local validation/test data and activate ML task**: option used to start the Master and load validation and test data
- **Display available end users**: shows a list of connected users and their addresses
- **Task alignment**: allows to estimate the alignment of the data of every contributing user with respect to the reference task defined by the master
• **Exclude unwanted users**: allows to exclude unwanted users, once the alignment is estimated

• **Ad Hoc local data pre-processing**: apply ad-hoc local pre-processing (possibly provided by the user) on the users’ data

• **Global data normalization**: estimate global parameters to perform a global normalization

• **Global feature selection**: apply a feature selection process on all data, to identify the most valuable features

• **Data value estimation**: estimate the value of every users’ data

• **Model training** (RidgeRegression): train the machine learning model

• **Performance on local validation/test**: compute performance on local validation/test data

• **Performance on users training data**: compute performance on users’ data (training data)

• **Create ROC figures**: Draw ROC figures comparing the performance of the implemented models

• **Terminate all user nodes**: sends a message to all users, asking them to terminate

• **EXIT MUSKETEER Demonstrator**: finalizes the demonstrator

In the next sections we will go through all these options and comment the observed results.

### 6.2 Master node initialization

The first step is to load the data local to the Master Node, and activate the Master ML process (Option "0" in the UI):

![Figure 16 The Master Node is initiated](image)
We assume that the user that defines the task (the one that runs the master node), has some data to evaluate the goodness of fit of the resulting models (validation and test datasets). It can be observed how the Master loads the validation and test data using the Data Connector. It also activates an ML task (Linear Classifier, implemented using a Ridge Regression model). Then it creates a Finite State Machine (FSM) to control the operational flow and it also creates any other local variable needed during the operation.

### 6.3 Display available end users

The demonstrator is able to show all the connected users and their addresses (Option "1"):

![Figure 17 listing the connected users](image)

We observe that 5 users are connected, and their addresses are shown.

### 6.4 Task alignment

The task alignment estimation is started with option "2".
The master starts exchanging information with the worker nodes (correlation values) and a task alignment value is produced for every one of them. As expected, the alignment of users No. 1 and 4 is low, indicating that the quality of their data is not as expected (for the task at hand).

### 6.5 Excluding unwanted users

The next logical step is to exclude from the training process those users with an alignment below a given threshold. In Option “3” of the Menu, the system invites to introduce such threshold value, and the participants with alignment below that value are excluded (disconnected).
6.6 Ad Hoc local data pre-processing

Another operation that can be performed from the Master Node is to apply an "ad hoc" local pre-processing at every worker node. Under this category may fall any operation that can be applied to the raw data without reference from the other users’ data. We have implemented...
Some examples of such ad-hoc local pre-processing options, in the final platform new ones will be added and, ultimately, the final user is responsible for providing any additional ad-hoc local pre-processing object associated to his/her task.

![Local Pre-processing Menu](image1)

**Figure 22 Local Pre-processing Menu**

We see that several options are available in the demonstrator. All these procedures can be applied without knowledge from the other participants:

- A logarithmic transformation;
- Image Down-sampling by different factors;
- Local Data Normalization (zero mean, unit standard deviation);
- Deep Learning Feature extraction (AlexNet model from torchvision, as depicted below (Original Architecture Image from [Krizhevsky et al., 2012.]). The network transforms any input image into a vector of 1000 features.

![AlexNet network](image2)

**Figure 23 AlexNet network used for Deep Learning pre-processing**
We illustrate the result of applying some of the available methods, for instance, the application of the down-sampling operator:

![Figure 24 Applying the down-sample pre-processing](image)

The worker shows the new size of its local training data:

![Figure 25 Workers applying the pre-processing and showing the new data size](image)
The application of the Deep Learning Pre-processing yields:

Figure 26 Applying the Deep Learning Pre-processing

6.7 Global data normalization

Another group of pre-processing methods may need global parameters to be applied. For instance, if we want that the ensemble set of training patterns has zero mean and unit standard deviation, it is necessary to estimate the normalizing values taking into account all the data from all users. This is achieved in option "5" in the Menu:
6.8 Global feature selection

Another pre-processing operation that needs global information is feature selection. For illustration purposes we have implemented a Greedy Feature Selection approach based on the training of linear models (LGFS). It can be applied to the data using option "6" in the Menu. The user needs to enter the number of features to be selected (and retained).
After this operation, the Master Node identifies which are the most relevant features, and then the local and remote data is pre-processed to retain only those features.

6.9 Data value estimation

An operation related to the task alignment one is the data value estimation. Several approaches can be implemented. For illustration purposes we will only include here one method, but more approaches will be investigated during the project. Since all these algorithms are implemented as separate objects, it will be straightforward to select one or another during the final operation of the MUSKETEER platform.

The data value estimation is obtained using option "7" in the Menu:
We observe that the user receiving a larger reward is the user with address "2", which coincides with the user providing the largest amount of fair data.

6.10 Model training

Once the training data has been pre-processed or transformed using the mechanisms selected by the Data Scientist controlling the MUSKETEER platform, the predicting model training itself can be executed, option "8" in the Menu:
6.11 Performance on local validation/test

Once the model has been trained, it is possible to measure its performance on the validation and test local data, for model selection and comparison purposes. Option "9" in the Menu provides this facility. The results of different experiments are stored, to be compared later. The Area Under ROC Curve (AUC) for this particular experiment is shown:

![Figure 31 Performance on local data](image)

6.12 Performance on training data at every user

Although it is not strictly necessary, for illustration of a possible operation on encrypted data, we provide the option to evaluate the model on the training data at every worker (without revealing the model to the workers). Since this operation on encrypted data can be extremely slow, we are showing here the results of the model trained with only two input features, and hence the low performance. After executing option "10":

---

D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version

36
6.13 Create ROC figures

For comparison purposes among different pre-processing options and models, we can generate ROC curves showing all the executed experiments, with option "11". As a result two figures are saved, showing the performance on the validation and test data sets:
Figure 33 Comparative ROC curves for different solutions on the validation set
Figure 34 Comparative ROC curves for different solutions on the test set

We observe that the worst performance is obtained when only two features are retained (i.e., using as input to the model only two selected pixels from the images) (AUC = 0.82), and the best result is obtained when a Deep Learning pre-processing (AlexNet) is applied (AUC = 0.996). The same classification model (RidgeRegression) has been used in all cases, which reinforces the importance of an adequate pre-processing of the data. Besides some standard pre-processing techniques that can be supplied to the final users, the user defining the task is possibly the best qualified to provide the best suited pre-processing object for a task at hand.

6.14 Terminate all user nodes

Option "12" allows to terminate all users.

6.15 Exit MUSKETEER Demonstrator

Option "99" exits MUSKETEER.
7 Software documentation

The documentation of the software is provided in html format along with the code. The documentation has been generated with Sphinx\textsuperscript{8}, and it will be maintained and expanded as the software project grows. We include in what follows the main pages of that documentation.

---

\textsuperscript{8}\textit{sphinx-doc.org}
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version
Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version
**Data Connectors**

A data connector that loads data from a file.

```python
class Demonstrator.models.data_connectors.Load_From_File.Load_From_File(filename):
    Bases: object

    This class implements a data connector, that loads the data from a file. This connector is specific for this demonstrator example.

    Create a Load_From_File instance.

    Parameters:  
        filename (string) -- path + filename to the file containing the data

    get_data_Master()  
        Obtains validation and test data

        Parameters:  
            None --  
                • Xval (ndarray) -- 2-D array containing the validation patterns, one pattern per row  
                • yval (ndarray) -- 1-D array containing the validation targets, one target per row  
                • Xtst (ndarray) -- 2-D array containing the test patterns, one pattern per row  
                • ytst (ndarray) -- 1-D array containing the test targets, one target per row

    get_data_worker(kworker)  
        Obtains training data at a given worker

        Parameters:  
            kworker (integer) -- number of the worker

        Returns:  
            • Xtr (ndarray) -- 2-D array containing the training patterns, one pattern per row  
            • ytr (ndarray) -- 1-D array containing the training targets, one target per row
```
Master Node

Master node object

class Demonstrator.nodes.MasterNode

Parameters:

- Nworkers (integer) – number of workers initially associated to the task
- end_users_addresses (list of strings) – list of the addresses of the workers
- master_address (string) – address of the master node
- comms (comms object instance) – object providing communications
- dc (DataConnector object instance) – data connector to the data of master and workers
- logger (class:logging.Logger) – logging object instance
- verbose (boolean) – indicates if messages are print or not on screen

Evaluate_Master_data()

Evaluate the trained model on the validation and test data at the Master

Parameters: None –

Evaluate_workers()

Computes AUC at the workers given the encrypted model

Parameters: None –

Returns: AUCs – dict of AUC estimation, one value per worker

Return type: dict of floats, the worker address is the key

Obtain_DV_estimation(dv)

Execute the data value estimation in the dv object

Parameters: dv (Data_Value object) – data value estimation procedure

Returns: data_value – dict of data value estimation, one value per worker

Return type: dict of floats, the worker address is the key

adhoc_local_preprocessing(prep)

Receives a preprocessing object and applies it to the data

Parameters: prep (Local_Preprocessing object) – The preprocessing object to be applied to local validation/test data and training data in the workers
count_TR_patterns()
Count total number of training patterns

Parameters: None –
Returns: NP_training – Number of total training patterns
Return type: integer

create_model_Master(model_type, NF)
Create the model object to be used for training at the Master side. By now only one model is available, but the list will grow and we will be able to choose here among a wide variety of options.

Parameters: model_type (str) – Type of model to be used

display(message)
Print message to log file and display on screen if verbose=True

Parameters: message (str) – string message to be shown/logged

estimate_task_alignment(ta)
Compute the task alignment estimates

Parameters: ta (TaskAlignment object) – The TaskAlignment object to be used for the estimation

feature_selection(fs, NF)
Execute the feature selection process implemented in the fs object

Parameters: fs (FeatureSelection object) – Feature selection procedure to be applied
NF (integer) – Number of features to retain (excluding the bias)

Returns: features – list of indices to the selected features
Return type: list of integers

filter_users(th_value)
Exclude users below a given thresholds in the task alignment index

Parameters: th_value (float) – Threshold value to apply: users below this value are excluded from the training process

get_TR_squared_sum(x_mean)
Compute the squared sum of all patterns in the workers

Parameters: None –
Returns: Xsquared_sum_TR – 1-d Array containing the sum of all squared training patterns
Return type: ndarray

get_TR_sum()
Compute the sum of all patterns in the workers

Parameters: None –
Returns: Xsum_TR – 1-d Array containing the sum of all training patterns
Return type: ndarray
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version

get_global_mean_std()

Obtain the global mean and std parameters

Parameters: None –

Returns:
- x_mean (ndarray) – 1-D array containing the mean values
- x_std (ndarray) – 1-D array containing the standard deviation values

load_data()

Load data to be used for validation/testing. Currently we load data from a file but in the future any other option is possible.

Parameters: None –

terminate_workers(users_addressesTerminate)

Terminate selected workers

Parameters: users_addressesTerminate (list of strings) – List of addresses of workers that must be terminated. If the list is empty, all the workers will stop.

train_model()

Train the Machine Learning Model

Parameters: None –

Worker Node

Worker node object

class Demonstrator.nodes.WorkerNode.WorkerNode(kworker, Nworkers, comm, dc, logger, verbose=False)

Bases: object

This class represents the Worker Node

Create a WorkerNode instance.

Parameters:
- kworker (string) – id of this worker
- Nworkers (integer) – number of workers initially associated to the task
- comm (comm object instance) – object providing communications
- dc (DataConnector object instance) – data connector to the data of master and workers
- logger (class:logging.Logger) – logging object instance
- verbose (boolean) – indicates if messages are print or not on screen

create_model_worker(model_type, reg=0.0001)

Create the model object to be used for training at the worker side. By now only one model is available, but the list will grow and we will be able to choose here among a wide variety of options.
Ad Hoc Local Preprocessing

Log Transform

Logarithmic transformation of the features.

class Demonstrator.models.preprocessing.ad_hoc_preprocessing.LogTransform
    Bases: object

    This class implements a procedure to locally preprocess the training data such that raw data is transformed into a set of input features ready for the training process. The applied transformation is a logarithmic function.

    Create a LogTransform instance.

    Parameters: None –

transform(X)

    Apply the transformation to the input data

    Parameters: X (ndarray) – 2-D array containing the patterns, one pattern per row

    Returns: X – 2-D array containing the transformed patterns, one pattern per row

    Return type: ndarray
Downsampling

Downsampling features by a given factor.

class Demonstrator.models.preprocessing.ad_hoc_preprocessing.DownSample.DownSample
    Bases: object
    This class implements a procedure to locally preprocess the training data such that raw data is transformed into a set of input features ready for the training process. The features are low pass filtered and then downsampled by the given factor.

Create a DownSample instance.

Parameters: downsample_factor (integer) – The downsampling factor

transform(X)
    Apply the transformation to the input data

Parameters: X (ndarray) – 2-D array containing the patterns, one pattern per row
Returns: newX – 2-D array containing the transformed patterns, one pattern per row. The number of features is reduced by a factor of “downsample_factor”

Return type: ndarray

Normalization

Local data normalization to zero mean and unit standard deviation

class Demonstrator.models.preprocessing.ad_hoc_preprocessing.Normalization.Normalization
    Bases: object
    This class implements a procedure to locally preprocess the training data such that every feature is transformed to have zero mean and unit standard deviation.

Create a Normalization instance.

Parameters: • x_mean (ndarray) – mean value externally provided
• x_std (ndarray) – standard deviation value externally provided

fit(X)
    Estimate the normalization values.

Parameters: X (ndarray) – 2-D array containing the patterns, one pattern per row
Returns: scaler – scaler object, stored at self.scaler
Return type: StandardScaler object

transform(X)
    Apply the normalization to the input data

Parameters: X (ndarray) – 2-D array containing the patterns, one pattern per row
Returns: X – 2-D array containing the normalized patterns
Return type: ndarray
Deep Learning

Local data preprocessing with a Deep Learning model

class Demonstrator.models.preprocessing.ad_hoc_preprocessing.DeepLearning

Bases: object

This class implements a procedure to locally preprocess the training data such that raw data is transformed using a Deep Learning Neural Network, its output values become the new set of input features.

Create a DeepLearning instance.

transform$(X)$

Preprocessing function: Deep Learning with Alexnet

Parameters: $X$ (ndarray) – 2-D array containing the patterns, one pattern per row

Returns: $\text{newX}_b$ – 2-D array containing the Deep Learning features

Return type: ndarray

Task Alignment

Task alignment among different users

class Demonstrator.models.task_alignment.task_alignment_1.TaskAlignment

Bases: object

This class implements a procedure to estimate the alignment among data from different workers with respect to the data of reference, using as input the correlation matrices.

Create a TaskAlignment instance.

estimate_alignment$(R_r\_ref, R_r\_workers\_dict)$

Estimate the task alignment.

Parameters:

- $R_r\_ref$ (tuple of ndarray $[R_r\_ref, r\_ref]$) – $R_r\_ref$ is a 2-D array containing the self-correlation matrix of the validation dataset $r\_ref$ is a 1-D array containing the cross-correlation matrix of the validation dataset
- $R_r\_workers\_dict$ (tuple of dicts of ndarray $[R_r, r]$, for every worker) – $R_r$ is a 2-D array containing the self-correlation matrix of the training dataset at a worker $r$ is a 1-D array containing the cross-correlation matrix of the training dataset at a worker

Returns: $dv$ – dict containing the alignment estimation, the key is the user address

Return type: alignment
Data Value

data_value_1

Data value estimation from correlation matrices

class Demonstrator.models.data_value.data_value_1.DataValue
    Bases: object

This class implements a procedure to estimate the Data Value of the data coming from different workers with respect to the data of reference, using as input the correlation matrices

Create a DataValue instance.

Parameters: None –

`estimate(Rr_ref, Rr_workers_dict)`

Estimate the data value of every worker.

Parameters:

- `Rr_ref` (tuple of ndarray `R_ref, r_ref`) – `R_ref` is a 2-D array containing the selfcorrelation matrix of the validation dataset `r_ref` is a 1-D array containing the crosscorrelation matrix of the validation dataset

- `Rr_workers_dict` (tuple of dict of ndarray `R, r, for every worker`) – `R` is a 2-D array containing the selfcorrelation matrix of the training dataset at a worker `r` is a 1-D array containing the crosscorrelation matrix of the training dataset at a worker

Returns: `dv` – dict containing the data value estimation, the key is the user address

Return type: `dict`

Feature Selection

LinearGreedyFeatureSelection

Feature Selection by a brute force greedy approach using linear models

class Demonstrator.models.feature_selection.LinearGreedyFeatureSelection.LinearGreedyFeatureSelection
    Bases: object

This class implements a procedure to select the best input features using a greedy brute force approach based on linear models

Create a LinearGreedyFeatureSelection instance.

Parameters:

- `pos_selected` (list of integers) – sorted list of features by importance, useful to apply the feature selection without solving again the fit part (persistance)

- `fit(R, r, Xval_b, yval, NT, regularization)`

Obtain the list of features to be extracted, sorted by relevance
Parameters:
- \( R (\text{ndarray}) \) – 2-D array containing the self-correlation matrix
- \( r (\text{ndarray}) \) – 1-D array containing the cross-correlation matrix
- \( \text{Xval}_b (\text{ndarray}) \) – 2-D array containing the validation patterns
- \( \text{yval} (\text{ndarray}) \) – 1-D array containing the validation targets
- \( \text{NF} (\text{integer}) \) – Number of features to extract (excluding bias)
- \( \text{regularization} (\text{float}) \) – Regularization value to be used in the models

Returns: features – List of selected features, the first one is always 0, the bias, stored at self.features

Return type: list

transform(X)

Apply the feature selection to the data

Parameters: \( X (\text{ndarray}) \) – 2-D array containing the patterns

Returns: \( X_{fs} \) – 2-D array containing the patterns with selected features

Return type: \( \text{ndarray} \)

Ridge Regression Model

Ridge Regression model

class Demonstrator.models.RidgeRegression.RidgeRegression_Master(Nworkers, end_users_addresses, comms, logger, verbose=False, regul=0.001)

Bases: models.POM6_ML.POM6ML

This class implements the Ridge Regression model, run at Master node. It inherits from POM6ML.

Create a RidgeRegression_Master instance.

Parameters:
- \( \text{Nworkers} (\text{integer}) \) – number of workers initially associated to the task
- \( \text{end_users_addresses} (\text{list of strings}) \) – list of the addresses of the workers
- \( \text{comms} (\text{comms object instance}) \) – object providing communications
- \( \text{logger} (\text{class:logging.Logger}) \) – logging object instance
- \( \text{verbose} (\text{boolean}) \) – indicates if messages are print or not on screen

CheckNewPacket_master()

Checks if there is a new message in the Master queue
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version
D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms – Initial Version
8 Conclusions

In this deliverable (D4.2) we have presented preliminary algorithms for data pre-processing, normalization and task alignment and data value estimation approaches. As this is an initial version and most of the research is ahead of us and also many algorithms are still to be implemented during the next months, these preliminary versions have served to define a basic software structure that will be partly inherited by the final platform. The operation of the algorithms has been illustrated in the form of a fully operable software Demonstrator (within the demonstration conditions: given a particular dataset and supporting a limited number of options). In spite of only being a first Demonstrator, it includes important concepts and software components design to be further incorporated in the final MUSKETEER platform: pre-processing objects, data connectors, a communication library, Master and Worker nodes, etc. The behaviour of the Demonstrator has been illustrated in this document, where the results of every available option have been briefly described. The Demonstrator has been tested on Linux, Windows and macOS platforms, and instructions for installation and execution are also included.

9 References


[Sphinx] https://sphinx-doc.org
