D3.2 – Feature and Structure modeling, Structured Input/Output, Unsupervised Learning and Domain Adaptation (Accompanying Document for the Engineering Release)

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Status</th>
<th>Work Package</th>
<th>Deliverable Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3.2</td>
<td>Working/Draft/Final</td>
<td>WP 3</td>
<td>Other</td>
</tr>
<tr>
<td>Date of Delivery</td>
<td>31/10/2016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Responsible Unit</td>
<td>UNITN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Editors</td>
<td>Kateryna Tymoshenko (UNITN)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contributors</td>
<td>Kateryna Tymoshenko (UNITN), Olga Uryupina (UNITN), Alessandro Moschitti (UNITN), Daniele Bonadiman (UNITN), Iryna Haponchyk (UNITN), Andrea Passerini (UNITN), Fabrizio Granelli (UNITN), Lei Xu (IBM), Haytham Assem (IBM), Teodora Sandra Buda (IBM), Alassane Samba (Orange), Philippe Dooze (Orange), Yosra Ben Slimen (Orange), Sylvain Allio (Orange), Julien Jacques (Orange), Bo Zhu (UPM), Bruno Ordozgoiti (UPM), Sandra Gómez Canaval (UPM), Alberto Mozo (UPM), Marco Quartulli (Vicomtech), Angel Martin (Vicomtech), Noelia Oses (Vicomtech), Juan Odriozola (Vicomtech), Iñaki Arruabarrena (Vicomtech)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reviewers</td>
<td>Marius Corici (Fraunhofer)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Author (Unit)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>29/08/2016</td>
<td>Working</td>
<td>Olga Uryupina</td>
<td>Document template</td>
</tr>
<tr>
<td>0.2</td>
<td>12/10/2016</td>
<td>Working</td>
<td>All partners</td>
<td>Initial contributions added to the document</td>
</tr>
<tr>
<td>0.3</td>
<td>17/10/2016</td>
<td>Working</td>
<td>Kateryna Tymoshenko</td>
<td>Summary/Introduction/CSE/Conclusions sections added to the document</td>
</tr>
<tr>
<td>0.4</td>
<td>20/10/2016</td>
<td>Draft</td>
<td>All partners</td>
<td>Revisions of the initial contributions added to the document</td>
</tr>
<tr>
<td>0.5</td>
<td>27/10/2016</td>
<td>Draft</td>
<td>Kateryna Tymoshenko</td>
<td>Prepared the pre-final draft</td>
</tr>
<tr>
<td>0.6</td>
<td>30/10/2016</td>
<td>Draft</td>
<td>Kateryna Tymoshenko</td>
<td>Addressed the comments of the internal UNITN reviewers</td>
</tr>
<tr>
<td>0.7</td>
<td>31/10/2016</td>
<td>Draft</td>
<td>All partners</td>
<td>Addressed the comments of the internal CogNet reviewer</td>
</tr>
<tr>
<td>1.0</td>
<td>31/10/2016</td>
<td>Final</td>
<td>Kateryna Tymoshenko, all</td>
<td>Final version</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable (D3.2) presents the first prototype of the machine learning components implemented within the H2020 5G-CogNet project as a part of CogNet Smart Engine (CSE), and its respective documentation. The proposed prototype constitutes a multi-purpose and multi-mechanism machine learning engine to address the 5G networks’ management needs.

The choice of learning algorithms was motivated by the requirements of the scenarios defined in D2.1 to exhaustively illustrate the prospective challenges of 5G networks. The previous deliverable (D3.1 [1]) provides the theoretical background behind the majority of the components presented in this deliverable. Some components, however, address the needs, which were defined later in the course of the project; therefore, we refer the reader to the external materials for their theoretical groundings.

The components included into this deliverable enable:

- Automated machine learning model selection (TCDC)
- Feature selection (PICS, IterFS)
- Supervised (Spark-TK, NetSpark, LSSVM-SPARK) and unsupervised (PSCEG, Funco) machine learning tools
- Streaming-mode classification (streamCluster) and regression (Spark-streaming-linear-regression)
- Heuristic function optimization network management support (SAOptimizer)
- User throughput prediction (ML4MQ)

Given that 5G networks are expected to operate upon massive amounts of data, all components of CSE must be able to function in the Big Data setting. In the particular case of the CogNet project, we use Apache Spark to enable this functionality. All the CSE components either support Spark, or adding such support is foreseen during the second year of the project.
# Table of Contents

1. **Introduction** ................................................................. 10  
2. **CogNet Smart Engine (CSE)** ........................................... 12  
   2.1. CSE components in the CogNet architecture .................. 12  
   2.2. Technical Nature of CSE .............................................. 14  
3. **CSE components** ............................................................... 16  
   3.1. SPARK-TK: Machine Learning for Structural Input (UNITN) ... 16  
      3.1.1. Download and Installation ........................................ 17  
      3.1.2. Training ............................................................. 19  
      3.1.3. Deployment ......................................................... 20  
      3.1.4. SPARK support .................................................... 21  
      3.1.5. Development status .............................................. 23  
   3.2. NetSpark: Feature Engineering with DNNs (UNITN) ........... 24  
      3.2.1. Download and Installation ........................................ 24  
      3.2.2. Training ............................................................. 25  
      3.2.3. Deployment ......................................................... 26  
      3.2.3.1 Examples .......................................................... 26  
      3.2.4. Development Status .............................................. 28  
   3.3. LSSVM-SPARK: Structured Output with Latent Variables (UNITN) ... 28  
      3.3.1. Download and Installation ........................................ 29  
      3.3.2. Deployment ......................................................... 30  
      3.3.3. Development status .............................................. 32  
   3.4. PICS: Unsupervised feature selection on Spark (UPM) .......... 32  
      3.4.1. Download and Installation ........................................ 33  
      3.4.2. Deployment ......................................................... 33  
      3.4.3. Example ............................................................ 35  
      3.4.4. Development status .............................................. 36  
   3.5. IterFS: Deterministic feature selection for unsupervised learning (UPM) .......... 36  
      3.5.1. Download and Installation ........................................ 36  
      3.5.2. Deployment ......................................................... 37  
      3.5.3. Example ............................................................ 37  
      3.5.4. Development status .............................................. 38
3.6. PSCEG: an unbiased parallel subspace clustering algorithm using exact grids (UPM) 38
  3.6.1. Download and Installation ................................................................. 39
  3.6.2. Deployment .......................................................... 39
  3.6.3. Example .......................................................... 40
  3.6.4. Development status .......................................................... 40
3.7. Funco: Model-based co-clustering for functional data (Orange) ................. 40
  3.7.1. Download and Installation ................................................................. 41
  3.7.2. Deployment .......................................................... 41
  3.7.3. Development status .......................................................... 45
3.8. ML4MQ: Mobile User Throughput Prediction (Orange) .......................... 46
  3.8.1. Download and Installation ................................................................. 47
  3.8.2. Deployment .......................................................... 48
  3.8.3. Development status .......................................................... 49
3.9. TCDC: Model Recommendation for Supervised Learning (IBM) ............... 49
  3.9.1. Download and Installation ................................................................. 50
  3.9.2. Deployment .......................................................... 51
  3.9.3. SPARK support .......................................................... 52
  3.9.4. Development status .......................................................... 52
3.10. Distributed Application Performance Optimizer (Vicomtech) .............. 53
  3.10.1. streamCluster: Online classification module based on the Spark platform 54
    3.10.1.1 Download and Installation ................................................................. 56
    3.10.1.2 Deployment .......................................................... 56
    3.10.1.3 SPARK support .......................................................... 62
    3.10.1.4 Development status .......................................................... 62
  3.10.2. Spark-streaming-linear-regression: Online linear regression with Spark’s Streaming API .......................................................... 62
    3.10.2.1 Download and Installation ................................................................. 62
    3.10.2.2 Deployment .......................................................... 63
    3.10.2.3 SPARK support .......................................................... 65
    3.10.2.4 Development status .......................................................... 65
  3.10.3. SAOptimizer: Optimisation algorithm for 5G networks ............... 66
    3.10.3.1 Download and Installation ................................................................. 66
    3.10.3.2 Deployment .......................................................... 67
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.10.3.3 SPARK support</td>
<td>70</td>
</tr>
<tr>
<td>3.10.3.4 Development status</td>
<td>70</td>
</tr>
<tr>
<td>3.11. Summary</td>
<td>73</td>
</tr>
<tr>
<td>4. Conclusions</td>
<td>74</td>
</tr>
<tr>
<td>References</td>
<td>75</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1 CogNet Smart Engine (CSE) architecture ........................................... 13
Figure 2 Output of svm_learn command .......................................................... 19
Figure 3 Running svm_classify on a toy test dataset ...................................... 20
Figure 4 Output of svm_classify on a toy dataset ........................................... 21
Figure 5 Output of Spark-TK classification on the toy dataset ........................... 23
Figure 6 The output after running PICS on a toy data set ............................... 35
Figure 7 The output after running IterFS on a toy data set ............................. 37
Figure 8 The result of running PSCEG on a toy data set. ................................. 40
Figure 9 Example of Functional Data ............................................................ 42
Figure 10. Running Functional co-clustering on a toy dataset .......................... 45
Figure 11. Plots of the parameters convergency test on a toy dataset ................. 45
Figure 12: Results on validation dataset ....................................................... 47
Figure 13 ML4MQ output ............................................................................... 49
Figure 14 TCDC Model Selection .................................................................. 52
Figure 15 Performance of Candidates ............................................................. 52
Figure 16 System block diagram .................................................................... 54
Figure 17 Classification module, block diagram ............................................. 55
Figure 18 Set up and trace of training data server ........................................... 60
Figure 19 Set up of the training data server controller ..................................... 61
Figure 20 Expected results at the output port ............................................... 61
Figure 21 SAOptimizer convergence at runtime .......................................... 68
Figure 22 Minimization of the sin function ..................................................... 69
Figure 23 Minimization of the Ackley function .............................................. 69
Figure 24 Minimization of the Bukin function N. 6 ....................................... 71
Figure 25 Maximization of the network connectivity ..................................... 71
Figure 26 Log of the evolution of the optimal result ...................................... 72
Figure 27 Evolution of the optimal result ....................................................... 72
List of Tables

Table 1 Technical details of CSE components in the current release..................................15
Table 2 Code snippets for running prepared scripts..........................................................59
1. Introduction

The prospective 5G networks will provide service to myriads of diversified devices that constantly exchange massive volumes of data according to their communication needs for a plethora of various services. In order to function smoothly without interrupting provision of their services, while being at the same time cost-efficient, and secure, they will require effective and highly responsive management techniques, adapted as much as possible to the specific situation. Manual management of these networks by means of a set of rigid predefined rules or policies does not scale. Therefore, the prospective 5G networks should be capable of highly adaptive cognitive self-management.

CogNet project focuses on enabling the situation-specific self-management in the networks through usage of machine learning. CogNet architecture, devised in Work Package 2, foresees that a single block called CogNet Smart Engine (CSE) will employ machine learning techniques to analyse the current state of the network and to provide a number predictions to be used in the policies when further changing state of the network.

This document contains the accompanying documentation for the first software release of CSE. This release contains the implementations of machine learning algorithms selected for CSE in Deliverable 3.1 [1]. Technologies contained within CSE will serve as instruments for Work Packages 4 and 5 when developing solutions for their scenarios. Since CSE must be able to efficiently work in the Big Data setting, each of its components is expected to support the Apache Spark framework. The majority of the components in this release already do so, while the remainder will support Spark in the next release (D3.3).

CSE currently contains tools for feature selection, supervised and unsupervised machine learning, real time classification and regression, and, finally, the appropriate machine learning model selection. All the tools are publicly available as open source. The theory behind most of the tools delivered within Deliverable 3.2 has been described thoroughly in Deliverable 3.1, with the following exceptions:

- Deterministic feature selection module, IterFS. Its theoretical background is described in [2].
- Model selection module, TCDC. Its theoretical background is described in [3].
- Heuristic optimization module, SAOptimization. Its theoretical background is described in [4], [5], [6], [7].
- Mobile user throughput prediction module, ML4MQ, which is a product of an ongoing research in Orange Labs, Lannion, France.
- Co-clustering for functional data, Funco. Its theoretical background is described in [8].

The technologies released within Deliverable D3.2 reflect research and development activities across all the CogNet WP3 tasks. Moreover, these technologies have been selected in collaboration with WP2, WP4 and WP5 to ensure the applicability of our Machine Learning research to practical 5G tasks. More details on the potential application of each module are provided in individual descriptions (Section 3).
The remainder of the document is structured as follows. Section 2 provides a high-level overview of CSE, its relation to the other CogNet architectural components and Work Packages, and the general technical details regarding the release. Section 3 contains the supporting documentation for all the CSE software components, providing instructions for the components’ installation, deployment, training new models (if applicable), and instructions for running an experiment on the toy datasets supplied along with the current software release. Finally, Section 4 provides a conclusion and the plans for the next WP3 software release.
2. CogNet Smart Engine (CSE)

5th generation (5G) mobile networks are the next phase in the development of telecommunication networks to be delivered by 2020. The main challenge to be addressed by the 5G networks consists in empowering cost efficient machine-to-machine interaction, otherwise known as the Internet of Things (IoT). IoT will consist not only of the smartphones inter-exchanging data, but also of the myriads of other devices, e.g. cars, domestic animals’ biochips, household appliances and other so-called “things”. Therefore, 5G networks should ensure high broadband, route extremely large amounts of data that circulate between billions of devices with the different broadband needs. They must provide low latency, ensure security, cost-effectiveness, elasticity and energy efficiency. The networks must be also able to promptly react to various events, such as a failing node or a rapid increase of the demand for service in a certain location.

It is infeasible to address the above tasks on the large scale by using single policy engines, as the devices diversify and so do the requirements. This raises a need for a management mechanism, which can automatically address the above challenges.

2.1. CSE components in the CogNet architecture

Current CogNet architecture devised in Work Package 2 addresses the challenges of self-management and dynamicity by adopting a Monitoring-Analysis-Planning and Execution (MAPE) Loop. MAPE architecture contains four phases in a continuous loop: monitoring phase collects information about the current state of the network and passes it to analysis phase. The latter processes information and sends its output, that can be either a value of a certain threshold (e.g. the minimal amount of connected devices before a network node failure) or some real-valued number (e.g. the expected amount of traffic in the next hours) to the planning phase. The planning phase uses the received information to devise the further plan of actions to increase/maintain the Quality of Service of the network, which, finally, is executed in the execution phase. Here, the analysis phase is performed by a set of machine learning techniques. Provided that data coming into analysis phase can be highly heterogeneous and given the diversity of the 5G challenges one cannot choose only one suitable machine learning technique, but must rely a set of Machine Learning algorithms for addressing different kinds of tasks.

Within CogNet project, we call such collection of machine learning algorithms CogNet Smart Engine (CSE). In the high level overview of CogNet architecture solution, defined in Work Package 2, this is conceptually a single block. It receives the characteristics of the current state of the network from the Data Collector component, analyses them and passes the results to the Policy Engine that uses the CSE output to propose appropriate network policies.

The internal architecture of the CSE itself, devised in Work Package 2, was inspired by lambda architecture, a framework designed to process large amounts of data in the batch and (near) real time modes. The CSE internal architecture is depicted in Figure 1. Here, the “Data pre-processing”, “Automated model selection”, “Batch
processing Engine" and "(Near) Real-time processing Engine" blocks heavily rely on machine learning. As evident from its name, the "Data pre-processing" unit processes the input coming from the network. The tasks that it performs are both technical, i.e. data normalization and transformation, and analysis-related, i.e. machine learning feature extraction and selection. The latter is very important, as one must select from the myriads of signals coming for the network, those that are useful for specific tasks. Designing good features is a laborious task for human experts, therefore in CSE, we power with block with automatic feature selection techniques, such as deep neural networks, NETSpark (Section 3.2) and algorithms based on random sampling, PICS and IterFS (Section 3.4, 3.5). "Automated model selection" is an optional block that proposes an appropriate machine learning model for the incoming data and task. In this code deliverable it is represented by the component for model recommendation for supervised learning, TCDC (train, compare, decide and change), described in Section 3.9. It picks a model which performs well on the input task, and is at the same time the simplest of all the possible models in terms of computational complexity, easiness of interpretability and implementation.

"(Near) Real-time processing Engine" and "Batch processing" blocks are main components which will employ both supervised and unsupervised machine learning models. The unsupervised models are represented by an unbiased parallel subspace clustering tool, (PSCEG, Section 3.6), and a module capable of clustering curves or functions (Funco, Section 3.7). The CSE supervised ML components can provide real-valued predictions on structured input data (Spark-TK, Section 3.1). Moreover, their predictions can also be a structure (LSSVM-SPARK, Section 3.3). Then, the set of supervised models includes the deep neural network module, which uses complex feature representations to provide a prediction (NetSpark, Section 3.2). Moreover, apart from the general-purpose models, CSE also already proposes a model trained for user

---

1 The figure was originally presented in the Year 1 report of Work Package 2

2 [http://disi.unitn.it/moschitti/Tree-Kernel.htm](http://disi.unitn.it/moschitti/Tree-Kernel.htm)
throughput prediction (ML4MQ, Section 3.8), valuable for the SLA Enforcement Scenario in Work Package 4. Then, CSE already provides the online machine learning capabilities useful both for classification and regression (streamCluster and Spark-Streaming-linear-regression, Sections 3.10.1, 3.10.2), particularly relevant for the “(Near) Real-time processing Engine” architecture component. Finally, the CSE contains a tool for function optimization (SAOptimizer, Section 3.10.3), for minimization in both discreet and continuous spaces with and without constraints. It will be useful, for example, for optimizing the topology of the network. Moreover, the streamCluster, Spark-Streaming-linear-regression and the SAOptimizer modules together constitute parts of the distributed application performance optimization framework described in Section 3.10. This framework aims to understand the status of the network (streamCluster), forecast the key performance indicators (Spark-streaming-linear-regression) and to propose the optimum arrangements for the further functioning of the network (SAOptimizer).

Note, that all the components, despite their different complementary purposes, will be deployed on the same cluster-computing framework, Apache Spark. Therefore, technically, CSE is a block with the single Big Data management interface, compatible with the remainder of the project components.

### 2.2. Technical Nature of CSE

We make the CSE components available in a CogNet project GitHub repository, [https://github.com/CogNet-5GPPP/WP3-CSE](https://github.com/CogNet-5GPPP/WP3-CSE). We place each CSE component into a separate folder, eponymous with the component’s name.

Table 1 reports technical details regarding the modules, such as their language of implementation, current statues of spark support and the Work Package 3 (WP3) task they refer to. All components run under Linux.

<table>
<thead>
<tr>
<th>Name (Developer)</th>
<th>Purpose</th>
<th>Language</th>
<th>Spark support</th>
<th>WP3 task</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCDC (IBM)</td>
<td>Model Selection</td>
<td>R</td>
<td>Under study for D3.3</td>
<td>T3.4</td>
</tr>
<tr>
<td>PICS (UPM)</td>
<td>Unsupervised feature selection</td>
<td>Java</td>
<td>Present</td>
<td>T3.1</td>
</tr>
<tr>
<td>IterFS (UPM)</td>
<td>Unsupervised feature selection</td>
<td>Python</td>
<td>Planned for D3.3</td>
<td>T3.1</td>
</tr>
<tr>
<td>NETSpark (UNITN)</td>
<td>Deep learning: feature selection and supervised learning</td>
<td>Python</td>
<td>Classification, planned for learning for D3.3</td>
<td>T3.1, T3.2</td>
</tr>
<tr>
<td>Spark-TK (UNITN)</td>
<td>Supervised learning on structured input</td>
<td>Java</td>
<td>Classification, planned for learning for</td>
<td>T3.2</td>
</tr>
<tr>
<td>CSE Component</td>
<td>Description</td>
<td>Programming Language</td>
<td>Status</td>
<td>Phase</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>-----------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>LSSVM-SPARK (UNITN)</td>
<td>Supervised learning with structured output</td>
<td>Java</td>
<td>Classification, planned for learning</td>
<td>D3.2</td>
</tr>
<tr>
<td>Funco (Orange)</td>
<td>Curve/function co-clustering</td>
<td>R</td>
<td>Planned for D3.3</td>
<td>T3.3</td>
</tr>
<tr>
<td>PSCEG (UPM)</td>
<td>Unsupervised unbiased parallel subspace clustering</td>
<td>Python</td>
<td>Present</td>
<td>T3.3</td>
</tr>
<tr>
<td>streamCluster (Vicomtech)</td>
<td>Online classification</td>
<td>Python</td>
<td>Present</td>
<td>T3.3</td>
</tr>
<tr>
<td>Spark-streaming-linear-regression</td>
<td>Online regression</td>
<td>Python</td>
<td>Present</td>
<td>T3.4</td>
</tr>
<tr>
<td>SAOptimizer (Vicomtech)</td>
<td>Discrete/continuous functions optimization</td>
<td>Python</td>
<td>Under study for D3.3</td>
<td>T3.4</td>
</tr>
<tr>
<td>ML4MQ (Orange)</td>
<td>User throughput prediction</td>
<td>Python</td>
<td>Present</td>
<td>T3.1</td>
</tr>
</tbody>
</table>

Table 1 Technical details of CSE components in the current release
3. CSE components

This section provides the documentation for the first CSE software release components. All the components are located in the shared GitHub repository, https://github.com/CogNet-5GPPP/WP3-CSE, and must be run in the Linux environment.

The documentation subsections for all the components follow an uniform structure. Each subsection refers to one component and starts with the brief description of its purpose in general and in the CogNet context. Then, we provide download and installation instructions. We repeat some of the instructions, in each subsection, so that each subsection would be self-contained. Next, for all the supervised components we provide procedures for training ML algorithms. Then, we describe how to deploy the given module and how to run it on some toy datasets. All the toy datasets are included into the release. Finally, we report the future development status of each tool.

When delivering this software release, it is important to ensure that these components share the same parallelization platform, Apache Spark. Part of the components was developed using the Spark machine learning library, MLib, thus they support Spark by default. For the other components, we added Spark support and described it in the “Spark Support” subsections.

The roadmap of the names of the tools, their purposes, programming languages and the status of their Spark support can be found in Table 1, Section 2.2.

3.1. SPARK-TK: Machine Learning for Structural Input (UNITN)

In the 5G context, we are focusing on automatic network management. This requires machine learning for objects encoding different network states and configurations. To provide high-performance modelling of such complex objects, CogNet Task 3.2 focuses on developing machine learning algorithms to be run directly on structural representations. The first design of our structural input module (Spark-TK) has been presented in Deliverable 3.1. In what follows, we provide technical documentation for using the module.

The Spark-TK module extends the publicly available SVM-Light-TK\(^2\) ML toolkit. This toolkit allows the user to learn SVM classifiers over structured inputs. Structured input may consist of trees and/or sequences, besides feature vectors. The SVM-Light-TK tool performs classification by taking a classification model as input and measuring the similarity between training and test instances. This similarity is computed by using the Tree Kernel measure, which has been proved useful in many tasks. We refer the reader to Deliverable 3.1 for more information on structural input learning. Within the CogNet project, we have implemented the following extension to the SVM-Light-TK toolkit:

\(^2\) [http://disi.unitn.it/moschitti/Tree-Kernel.htm](http://disi.unitn.it/moschitti/Tree-Kernel.htm)
• the SPARK support has been integrated to the original package. This increases the scalability, making the structured input classifiers usable in BigData scenarios and, more specifically, for data-intensive 5G applications

Within the CogNet project, we will use SPARK-TK for modelling complex input object. In particular, we envisage experiments on encoding network topology with structural kernels. We are also relying on tree kernels within the Situational Context use-case to provide high-quality representation of societal context that might affect network consumption patterns.

3.1.1. Download and Installation
Spark-TK requires the installation of the SVM-Light-TK ML tool. Follow the instructions below to install the SVM-Light-TK tool.

1. Clone Spark-TK from the repository

Use the following command to clone Spark-TK from the repository.

```bash
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
```

Then, go into the project directory.

```bash
$ cd ./WP3-CSE/Spark-TK
```

2. Install Apache Spark™

Download the pre-build version of Spark from the Apache website

```bash
$ wget http://d3kbcqa49mib13.cloudfront.net/spark-2.0.0-bin-hadoop2.7.tgz
```

Decompress the zip archive

```bash
$ tar -zxvf spark-2.0.0-bin-hadoop2.7.tgz
```

Set the SPARK_HOME environment variable:

```bash
$ export SPARK_HOME=path/to/unpacked/spark/distribution
```

3. Set up the SVM-Light-TK tool

First you need to compile the SVM-Light-TK v1.5 shared library. This library is used by the SVM-Light-TK tool with SPARK support (see Section 3.1.4). Go into the project directory.

```bash
$ cd /dir/to/project/Spark-TK
```

Go into the SVM-Light-TK-1.5.Lib software directory

```bash
$ cd ./tools/SVM-Light-TK-1.5.Lib
```

Build the project by using the `make` command. Important: be sure to correctly set your `JAVA_HOME` environment variable and add the `JAVA_HOME/bin` directory to the `PATH` environment variable.
$ export JAVA_HOME=/path/to/jvm/bin  
$ export PATH=PATH:$JAVA_HOME/bin  
$ make clean libsvmlight_tk.so

Next, you need to package the SVM-Light-TK classifier into a jar. Go back to the project directory.

$ cd /dir/to/project/Spark-TK

Compile the SVM-Light-TK Java wrapper

$ javac tools/SVM-Light-TK-1.5.Lib/svmlighttk/SVMLightTK.java

Package into a jar

$ cd tools/SVM-Light-TK-1.5.Lib  
$ jar cf svmlighttk.jar svmlighttk  
$ cd ../ ../  
$ mkdir SVMClassify/lib  
$ cp tools/SVM-Light-TK-1.5.Lib/svmlighttk.jar SVMClassify/lib/

4. **Compile the Spark-TK scala app**

This step requires **sbt**, the build tool for Scala. If you do not have have **sbt** installed on your system, please refer to the following page for instructions on how to install **sbt**: [http://www.scala-sbt.org/0.13/docs/Manual-Installation.html](http://www.scala-sbt.org/0.13/docs/Manual-Installation.html). Please note that Scala works with Java 1.8+.

Go to the project folder

$ cd dir/to/project/Spark-TK

Go into the Scala project directory

$ cd SVMClassify

Compile the Scala project

$ sbt clean compile package

5. **Unpack the toy dataset (optional)**

If you plan to run experiments on the toy dataset supplied with the tools, you need to unpack them first:

$ cd dir/to/project/Spark-TK
$ unzip data/examples/toy-dataset.zip -d data/examples/
3.1.2. Training

The input format of Spark-TK training files is the same as that of SVM Light TK, described in detail in http://disi.unitn.it/moschitti/Tree-Kernel.htm.

The SVM-Light-TK tool is located in the tools directory within the project directory: path/to/project/Spark-TK/tools/SVM-Light-1.5-rer. You must build it before using:

```
cd path/to/project/Spark-TK/tools/SVM-Light-1.5-rer
make clean
make
```

The command for training a model by using the SVM-Light-TK v1.5 software is:

```
$ svm_learn [options] example_file model_file
```

Here, the arguments are as follows:

- **example_file** -> file with training data
- **model_file** -> file to store the learned SVM-Light-TK v1.5 model
- **[options]** -> Options that can be used with the SVM-Light-TK v1.5 are listed in http://disi.unitn.it/moschitti/Tree-Kernel.htm

Here is an example on how to learn a new model on a toy dataset.

```
$ cd dir/to/project/Spark-TK
$ ./tools/SVM-Light-1.5-rer/svm_learn -t 5 -F 2 -C + -W R -V R -m 1000
data/examples/toy-dataset/svm.train data/examples/toy-dataset/svm.model
```

Figure 2 Output of svm_learn command
Figure 2 demonstrates the console output of the previous command. The output reports the number of training examples read in memory, the number of support vectors in the final model and the training time in cpu-seconds.

### 3.1.3. Deployment

The command for classifying new examples with a pre-trained model by using the SVM-Light-TK v1.5 software is the following:

\[ \texttt{svm\_classify [options] example\_file model\_file output\_file} \]

Here, the arguments are as follows:

- **[options]** -> Options that can be used with the SVM-Light-TK v1.5 are listed in [http://disi.unitn.it/moschitti/Tree-Kernel.htm](http://disi.unitn.it/moschitti/Tree-Kernel.htm).
- **example\_file** -> file with test data in SVM-Light-TK v1.5 format
- **model\_file** -> file with the pre-trained SVM-Light-TK v1.5 model
- **output\_file** -> file where to store the predictions produced by the pre-trained model

Here is a toy example on how to do classification with the model we trained in subsection 3.1.2:

\[
\texttt{cd dir/to/project/Spark-TK}
\texttt{./tools/SVM-Light-1.5-rer/svm\_classify data/examples/toy-dataset/svm.test data/examples/toy-dataset/svm.model data/examples/toy-dataset/svm.pred}
\]

Figure 3 demonstrates the console output of the previous command. The output reports the general amount of support vectors in the model and the prediction time in cpu-seconds. Finally, in this specific examples testing the model on a test set for which we know the gold standard labels, and these labels are included into the test file in SVM-Light-TK v1.5 format. In this case, the tool reports standard machine learning classification performance metrics, such as accuracy, precision, recall and F1-measure.

Figure 3 Running svm\_classify on a toy test dataset

Figure 4 shows the content of the first lines of the file with the predictions. It is aligned line-by-line with the respective test file. SVM-Light-TK v1.5 performs binary classification, i.e. it assigns an example to one of two classes, positive or negative. Each line in the prediction file reports distance to the decision hyperplane for a corresponding machine learning example. If the distance is greater than zero, then the respective example is classified as belonging to the positive class, otherwise it belongs to the negative class. In Figure 4, some examples are classified as negatives (1st, 2nd, 3rd, 6th and 9th), while the other are classified as positives (4th, 5th, 7th, 8th and 10th).
3.1.4. SPARK support

Running predictions on the real-life datasets which may contain hundreds gigabytes of data might be time consuming. Adding SPARK support to the `svm_classify` module allows us to parallelize the prediction process, thus considerably speeding it up.

The command for running SVM-Light-TK classifier v1.5 with SPARK support is shown below. This command must be run in the Spark-TK project directory.

**Important:** Please ensure that shared library libsvmlight_tk.so has been compiled and is present in the project folder (e.g. tools/SVM-Light-TK-1.5.Lib/libsvmlight_tk.so).

```bash
$ SPARK_JAVA_OPTS="-Xss1g" $SPARK_HOME/bin/spark-submit \
  --driver-library-path "tools/SVM-Light-TK-1.5.Lib" \
  --class "SVMClassify" \
  --master local[*] \
  --jars svmlighttk.jar \
  SVMClassify/target/scala-<scala_version>/svmclassify-project_<scala_version>-1.0.jar \
  example_file model_file output_file num_partitions
```

JVM/Spark configuration arguments:

- `-Xss1g` -> set the initial and minimum Java heap size to 1 Gb of memory for each thread used by Spark-TK.
- `--master local[*]` -> run the code with the maximum available number of threads.
- `--class SVMClassify` -> run the SVMClassify scala class contained in the svmclassify-project_2.11-1.0.jar, which runs svm_classify in the multi-thread mode.
- `--driver-library-path "tools/SVM-Light-TK-1.5.Lib"` -> specifies the location of the directory containing the libsvmlight_tk.so shared library (required).
- `--jars svmlighttk.jar` -> add the SVMLightTK java library to the jar dependencies.
• `<scala_version>` -> scala version number, e.g. if you are using Scala 2.11.x, this parameter should be set to 2.11. Watch the output of the Step 4 of the installation instructions (Section 3.1.1) for the exact value.

Program Arguments:

• `example_file` -> file with the test data in the standard SVM-Light-TK v1.5 format
• `model_file` -> file to retrieve the learned decision rules
• `output_file` -> file where to store the predictions
• `num_partitions` -> the number of partitions in which to cut the dataset into

As first thing, Spark is initialized with the supplied Spark Configuration arguments. After that, Spark creates a distributed dataset containing the test examples to classify. The dataset is divided into a certain number of partitions. The parameter `num_partitions` specifies the number of partitions to split the dataset into.

Once the test set has been split, Spark will run one task (e.g. classification) for each partition of the dataset. The results of classification will later be written into the output directory named `output_file`. The output directory will contain many files named part-00000, part-00001, part-00002, etc... Typically, the number of files generated corresponds to the number of partitions passed in input by using the parameter `num_partitions`.

The command to launch Spark on the toy dataset (with Scala 2.11.x) is as follows:

```
$ SPARK_JAVA_OPTS="-Xss1g" $SPARK_HOME/bin/spark-submit --driver-library-path "tools/SVM-Light-TK-1.5.Lib" --class "SVMClassify" --master local[*] --jars svmlighttk.jar SVMClassify/target/scala-2.11/svmclassify-project_2.11-1.0.jar data/examples/toy-dataset/svm.test data/examples/toy-dataset/svm.model data/examples/toy-dataset/svm.pred 10
```

**Spark-TK.sh**

The software distributed in this work package comes with a handy shell script, named Spark-TK.sh, which launches Spark-TK with same default parameters listed above. Furthermore, it takes care of merging all the files generated by Spark-TK (e.g. part-00000, part-00001, etc.) in one single output file.

To run this script with default parameters, go to the project directory and type the following command:

```
$ ./Spark-TK.sh example_file model_file output_file scala_version num_partitions
```
3.2 Feature and Structure modeling, Structured Input/Output, Unsupervised Learning and Domain Adaptation

Figure 5 Output of Spark-TK classification on the toy dataset

The predictions generated by Spark-TK are written in the file named `output_file`. `scala_version` is scala version number, e.g. if you are using Scala 2.11.x, this parameter should be set to 2.11.

To run the command above on the toy dataset, with Scala 2.11.x and ten classification tasks to be run in parallel, type:

```
./Spark-TK.sh data/examples/toy-dataset/svm.test data/examples/toy-dataset/svm.model data/examples/scala-pred 10 2.11
```

Figure 5 shows the output printed to the console by Spark-TK when performing parallel classification of the test examples in the toy dataset. As can be seen, the execution time of Spark-TK – launched with 10 classification tasks in parallel – to classify all the examples in the test set, is considerably reduced with respect to the time needed by SVM-Light-TK (22.02 seconds in Figure 5 v.s. 170.75 in Figure 3).

3.1.5. Development status

As of today, the SPARK support has been correctly integrated into the SVM-Light-TK software. In the second release of this software, we would like to test it in many more applications and, eventually, conceive new features/kernels to be plugged into the Spark-TK ML tool for solving 5G-related tasks.
3.2. NetSpark: Feature Engineering with DNNs (UNITN)

Machine learning algorithms for the 5G context require a lot of task specific feature engineering. Neural Networks (NNs) are able to automatically extract complex feature representation from the raw data and consequently to train end-to-end classifier in complex domain.

The initial design for feature engineering with NNs and deep NNs (DNNs) was presented in Deliverable 3.1, Section 3.1. A neural network takes the dense vectors of features as input; then, in a general scenario, it recomposes and transforms these features by passing them from hidden layers to hidden layers until its final output layer outputs the prediction for the original example.

NNs, in particular, very deep NNs, (DNNs), may create bottlenecks in a production environment when processing high amount of data, in particular from data streams. Therefore, in this deliverable, we present the NetSpark module, an extension of the publicly available Keras\(^3\) library. NetSpark enables high performance feature engineering and classification with DNNs by extending the scalability of Keras’ usage in a production environment.

More specifically, within the CogNet project we have integrated Spark support into the original the Keras library, thus increasing its scalability in a production environment such as data-heavy 5G applications. In the current stage of development, we have enabled the deployment of the Keras pre-trained NN models over the Spark infrastructure.

NetSpark enables DNNs to be used in a number of scenarios, for example, in the large-scale event scenario, where it can process both 5G network-internal data (i.e. traffic analysis) and 5G network-external data (i.e. social media streams).

3.2.1. Download and Installation

The NetSpark module is built in Python and requires the following libraries to be installed on the system.

```
| Python 2.7 |
| Pip |
| Hdf5 |
| Spark >= 1.5 |
```

1. **Clone NetSpark from the repository**

Use the following command to clone the NetSpark repository.

```
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./WP3-CSE/NetSpark
```

\(^3\) [https://keras.io/](https://keras.io/)
2. **Setup and install NetSpark**

To install NetSpark run the following commands:

```bash
$ cd dir/to/project/NetSpark
```

Install the library using the python package manager (pip)

```bash
$ pip install .
```

This will install all the Python dependencies to run the library (i.e. Keras, Theano, h5py) as well as the library itself.

### 3.2.2. Training

At the current stage of development, NetSpark does not support distributed training on Spark. However, it is possible to train Neural Networks (NNs) locally using the Keras module.

A python script should be defined to train a new model. Such script should provide a definition of the architecture and of the adopted training strategy (i.e. optimization algorithm). These components are task-dependent, therefore, we cannot include a general-purpose command line script into the NetSpark library for launching training on a new task. More in detail, the training process is carried out at three different stages described by a simple example. The latter can be found in the example folder of the NetSpark library in `examples/toy/train.py`.

1. **Specify the architecture**

In order to define a new neural network architecture a sequential model should be created along with the various layers to be added to it as follows:

```python
model = Sequential()
model.add(Dense(4, activation='tanh', input_shape=(2,)))
model.add(Dense(1, activation='sigmoid'))
```

This network takes two features as input and outputs a score in the range [0, 1]. Once the model has been defined, it needs to be compiled with the proper optimizer and loss.

```python
model.compile(optimizer='adadelta', loss='binary_crossentropy')
```

2. **Train the model**

To train the model it is enough to call the `fit` function:

```python
model.fit(X_train, y_train, nb_epoch=10)
```

Here, `X_train` is a Numpy array containing the training examples, and `y_train` contains the label.

`Nb_epoch` is the number of iterations (epochs) over the data.

3. **Save the trained model**
The trained model can be saved using the `save` API:

```python
model.save('PATH_TO_THE_SAVED_MODEL')
```

For additional details on how to define and train Neural Network models please refer to the Keras documentation at: https://keras.io/

### 3.2.3 Deployment

Our contribution, at the current stage of development, is to enable deployment of the Keras pre-trained NN models over the Spark infrastructure. To do that, you need to create a `pyspark` script and import the `spark` module of `netspark` into it.

```python
from netspark import spark
```

This module provides an API to load any trained model into the spark context.

```python
conf = SparkConf()
sc = SparkContext(conf=conf)
model = spark.get_model(sc, 'PATH_TO_THE_SAVED_MODEL')
```

The loaded model has APIs to perform prediction and classification (`model.predict`, `model.classify`) using the spark map or map partition API.

In the following example Spark listens to a socket and reads the incoming string examples with tab-delimited features. It converts each example into a numpy array and classifies it into a specific class using the pre-trained NNs model.

```python
output = ssc.socketTextStream("localhost", 9999)\
        .map(lambda s: np.fromstring(s, dtype=np.float32, sep="\t"))\
        .map(model.classify)
```

### 3.2.3.1 Examples

The NetSpark directory contains two working examples: NetCla: The ECML-PKDD Network Traffic Classification Challenge, and a Toy network that performs the binary XOR.

**NetCla: The ECML-PKDD Network Traffic Classification Challenge**

The NetCla script loads a multi-layer perceptron trained on the data of the NetCla: The ECML-PKDD Network Traffic Classification Challenge⁴. This challenge was organized by CogNet at the ECML-PKDD 2016 conference. The participants were supposed to predict the type of application generating traffic on the network, using some data from sensor applied to the network. The data is described by machine learning features modelling the behaviour of the transmission over a specific network.

---

The provided script loads the test set and splits it in 10 different partitions. After loading and normalizing the data it classifies each example in the partitions using a Multi-Layer Perceptron, defined and pre-trained using the Keras library, and generates the submission file.

To run the script, execute the following commands from the root folder of the distribution.

```
$ cd data
$ unzip test_spark.csv.zip
$ cd ../examples
$ spark-submit --master 'local[4]' mlp_netcla.py
```

The submission file generated by the above script can be found in the data folder and is called `out_targ.csv`. The file contains the list of applications id; the value in each row represents the predicted application for each example in the `test_spark.csv` file.

Note that the model in this example has been compiled and trained with float32 weights, therefore you may need to change the Keras parameter `floatx` to float32 by editing the `~/.keras/keras.json` file.

**Toy XOR**

The Toy Network example tests the whole library (both training and test) on a self-contained task.

In this scenario, the input is a binary array of two features and the output label is the value of the xor function between the two inputs. In the `train.py` script the network is defined and trained on an input dataset that is generated programmatically. The trained network is then saved to a file.

```
$ cd dir/to/project/NetSpark/examples/toy
$ python train.py
```

To deploy and test the trained model there are two options: test from a text file and from a net socket stream.

To run test on the text file locally:

```
$ spark-submit --master 'local[4]' spark_test.py
```

This script predicts the “classes” of the input examples in `test_spark.csv` (i.e. the XOR value for the input binary variables) and writes them to `test_spark.csv.out`.

In order to run the model on the test data coming from a net socket stream, one needs to open a new Terminal window and launch a Netcat socket that generates a stream of input examples.

```
$ while sleep 1; do echo "1\t1"; done | nc -lk 9999
```

In another terminal window you can then launch the script that reads the above input stream and every minute provides the predictions for the examples received in input via Spark.
3.2.4. Development Status

Currently, NetSpark provides API to deploy Keras models into the Spark environments to classify and predict new examples using the `map` or `mapPartitions` Spark API.

In future releases our plan is to (i) extend the coverage of the Keras APIs to support distributed training of NNs models; (ii) Means to extract feature representations generated by the network at intermediate stage of computation (i.e the outputs of hidden layers).

3.3. LSSVM-SPARK: Structured Output with Latent Variables (UNITN)

The tasks arising in the automatic network management scenarios go far beyond the simple ML models and encompass myriads of interrelated parameters. As emphasized in Deliverable 3.1 [1], we intend to take advantage of the eventual complex (at times structural) interdependencies between parameters. More specifically, we plan to build models that predict their values simultaneously, in a form of a structure, or use this structural information at an intermediate stage as a latent factor.

More straightforwardly, there is a number of general structured output models from which the 5G networks can derive their benefit. First, networks are intrinsically variable in time. With the temporality aspect in consideration, the need might emerge for modeling sequence prediction, which inherently exemplifies a structured output problem, e.g., sequence of network states over time. Next, optimizing the network structure might benefit from ML models, which learn to predict such a complex object as a network structure graph. Finally, the task of clustering can be cast as a task of structured output prediction, and there is always a room for clustering techniques required both directly and for supplying auxiliary information to other tasks.

This contribution focuses on the latter, namely on applying structural predictions models for clustering. In the LSSVM-Spark module, we extend one of the instantiations of the Latent SVM\textsuperscript{struct5} (LSSVM) [9] previously referred to in Deliverable 3.1 [1], which is aimed for solving clustering prediction tasks. LSSVM takes a set of elements of arbitrary nature as an input example. The task consists in grouping the input example’s elements into clusters. Latent SVM\textsuperscript{struct} approaches it by introducing a latent auxiliary structure – a latent graph of pairwise relations between the elements of the example, and then by solving a spanning graph problem. The nodes of each connected component in the resulting spanning forest form a separate cluster. Our extension includes:

- providing SPARK support to the classification phase of the learner
- extension to a filtered setting (The original LSSVM implementation is designed for the case when, in the input data, the information is available about all the

\[\text{http://www.cs.cornell.edu/~cnyu/latentssvm/}\]
pairwise links between the elements. Our modification enables the use of partial data, when some pairwise links between graph nodes are absent.)

- enabling the use for inference of a greedy search in the latent graph (The greedy search is a restriction of the Chi-Liu-Edmonds’ [10], [11] MST algorithm to acyclic graphs [12], and it is faster than the Kruskal's [13] MST algorithm originally employed in LSSVM, which becomes crucial for large graphs.)

3.3.1. Download and Installation

This is a Maven project. You need Java 1.8.0_102 or later installed and configured. We have built the project under Spark 2.0 for Scala 2.1.

Use the following command to clone LSSVM-Spark from the repository.

```bash
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./WP3-CSE/svm-struct-latent-spark-classifier
```

The framework is an extension of the LSSVM written in C/C++, which methods (in lssvm-spark/) are called from Scala through JNI interface. Therefore, you, first, need to

1. **Provide a path to jni.h and jni_md.h**

Set up the JNILIB variable in lssvm-spark/Makefile. Modify the path if it is different on your machine.

2. **Download the original LSSVM library**

```bash
$ cd lssvm-spark/
$ wget http://www.cs.cornell.edu/~cnyu/latentssvm/latentnpcoref_v0.12.tar.gz
$ tar -xvf latentnpcoref_v0.12.tar.gz
```

add

```
#include <assert.h>
```

to latentnpcoref_v0.12/np_helper.cc, and add

```
#include <cstddef>
```

to latentnpcoref_v0.12/DisjointSets.cpp

Add -fPIC flag to CFLAGS in

- latentnpcoref_v0.12/Makefile
- latentnpcoref_v0.12/svm_light/Makefile

**Compile the native code of LSSVM-spark**

```bash
$ export JAVA_HOME=/path/to/jdk
$ make clean
$ make
```
3. Build the project

Return to the project folder

```bash
$ cd ..
```

You might need to configure Scala and Spark plugins, which is done in `pom.xml`
(http://stuq.nl/weblog/2008-11-26/4-steps-to-add-scala-to-your-maven-java-projects,
https://sparktutorials.github.io/2015/04/02/setting-up-a-spark-project-with-maven.html). And, you are ready to build the project:

```bash
$ mvn clean package
```

4. Unpack toy example data

If you plan to run the experiment on the toy example data supplied along with the
current tool, you need to unpack them first:

```bash
cd data
unzip lssvm-cluster1M.model.zip
unzip test.data.zip
cd ..
```

3.3.2. Deployment

Inside the project directory, the `target/` already contains a fat jar that can directly be
executed in a Spark environment.

```bash
$ cd ./target
$ spark-submit
   -class it.unitn.cognet.MainSparkClusterer
   --master local[*] 
   svm-struct-latent-spark-classifier-0.0.1.jar-with-dependencies.jar 
   data_file model_file output_directory_path
```

The arguments here are as follows:

- **data_file** – input file with the data to be classified
- **model_file** – file with a pre-trained model
- **output_directory_path** – path to the directory where to save the files with the output clustering results

With option **--master**, you need to specify a master URL for the Spark cluster. In our
case, **local[*]**, which is also a default value, means that we are running Spark locally
and using as many working treads as many logical cores we possess. In square brackets,
you can explicitly specify the desired number of threads, e.g., **--master local[4]**.
The **data_file** should comply with the SVM\textsuperscript{light} format, with each line denoting a feature vector starting with a label (‘1’ or ‘-1’) followed by a list of sparse features, e.g.,

```
-1 912:1.0 1295:1.0 1574:1.0 ... 997180:1.0 # 1 1 2
```

Such feature vector encodes information about the relationship between two elements of an example, and corresponds to one edge in the latent graph. The numbers after ‘#’ denote <example Id>, <1\textsuperscript{st} element Id> and <2\textsuperscript{nd} element Id>. This way, all the feature vectors with the same <example Id> belong to the same example. Label ‘1’ says that the corresponding elements belong to the same cluster, while label ‘-1’ says that they fall into different clusters. In other words, example’s data form a graph of pairwise links between nodes, each node corresponding to one element of the example.

The **model_file** should be of a form:

```
# sizePsi:1000000
1:2135.386245035224
2:-1733.249710086104
3:-1187.064549328849
4:-541.277273649726
.
.
1000000:-816.5227220015656
```

containing real-valued weights for the features.

While a model of the above form, being linear, in essence, is uniform over the application cases, a training phase depends on and is adapted to the task specifics up to a certain degree. The software for the training phase for producing such a model will be available in the future code releases.

On successful execution of the classifier, in the directory **output_directory_path**, one can find output files in the number of the classified examples, each named with the corresponding <example Id> and having a two-column format:

```
1 1
2 1
3 3
4 4
5 3
.
.
20 18
```

where the first column contains <element id> and the second – the <cluster id> to which the element was assigned by the classifier.

\[6\]  \url{http://svmlight.joachims.org/}
The current distribution contains a sample dataset and a pre-trained model in the `data/` directory. The classifier can be tested by running the following commands:

```
$ cd target/
$ rm ../data/cluster/*
$ spark-submit --class it.unitn.cognet.MainSparkClusterer --master local[*] 
  svm-struct-latent-spark-classifier-0.0.1-jar-with-dependencies.jar 
  ../data/test.data ../data/lssvm-cluster1M.model ../data/cluster
```

The clustering results will be saved in `data/cluster/`.

### 3.3.3. Development status

In the second release to this module, we plan to provide a complete structured prediction model, including also a training part with the Spark support along with the classification phase presented above. The type of the structured learning algorithm will be selected on the basis of the underlying task and the data structure. Running beforehand, most certainly, work will be required on the development/selection of suitable inference procedures.

### 3.4. PICS: Unsupervised feature selection on Spark (UPM)

Parallelized Independent Columns Selection (PICS) is an algorithm designed to take advantage of random sampling for column subset selection in distributed environments efficiently.

The algorithm is designed to find approximate solutions to the Column Subset Selection problem, that is, it attempts to find subsets of features that are the most representative of the whole data set in terms of linear approximations. This method provides feature subsets that can be interpreted in terms of simple linear models, a functionality that can be exploited in the field of network management and traffic analysis.

Feature selection is often instrumental for machine learning and data mining. Network traffic data usually come in very large volumes and labelling them is unfeasible. Unsupervised methods for feature selection can therefore be very helpful to network managers and engineers that want to perform successful exploratory analysis, visualization or unsupervised learning on their data. Inspired by results from the field of randomized linear algebra, we design a parallelized algorithm for column based matrix factorization that translates naturally to unsupervised feature selection. Its implementation on Apache Spark makes it applicable to huge datasets, taking advantage of distributed computing clusters.

PICS functions as follows: given an input data matrix it performs a judicious random sampling of candidate column subsets and returns the one that minimizes the loss of the approximation.
PICS is especially useful when large quantities of high dimensional data are available. This is the case of the Collaborative Resource Management use case, where large amounts of network traffic packets are being collected and aggregated into 5-tuple flows represented in over one hundred features. The output of PICS provides:

- Subsets of features that account for most of the variation in the data
- Models that reveal relationships between features.

This allows the user to obtain more manageable data sets for exploratory analysis and inference, as well as to infer relationships between the features beyond simple one-to-one correlation measurements.

### 3.4.1. Download and Installation

#### 1. Clone PICS from the repository

Use the following command to clone PICS from the repository.

```bash
$ git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
$ cd ./WP3-CSE/pics
```

#### 2. Compiling

PICS requires the installation of Maven 2.0. Instructions can be found at https://maven.apache.org/.

The project uses Maven to compile & build. To download the dependencies and create the required jar file in the target directory, type the following command from the main directory of the project:

```
$ mvn package
```

This will generate a new directory called target containing an executable jar file named pranks-1.0.jar. This jar file is a Spark executable that contains all the necessary dependencies.

### 3.4.2. Deployment

To run the algorithm, from the project path run the algorithm as indicated in the following synopsis:

```
$ spark-submit target/pics-1.0.jar --file=<input-file> --k=<integer> [--mp=<minPartitions> --scaling=BOUT|PR --sampling=TRAD|OWN --samples=<integer> --sv=<svectors> --svd=<TRUE|FALSE> --std=<TRUE|FALSE>]
```

- **input-file**: The relative path to the input file. Each line should contain the elements of the corresponding matrix row, separated by whitespace characters.
- **k**: the number of columns (features) to keep.
minPartitions: The minimum number of partitions for the input file, as specified by Spark.

sv: The number of singular vectors and values to consider. Default: k.

svd: whether or not to compute the SVD to compare the obtained residual Frobenius norm to that of the best rank-k approximation. If TRUE, said norm will be also written to the output. Default: FALSE.

std: whether or not to standardize the data to zero mean and unit variance. Default: TRUE.

scaling: The approach for scaling the input of the RRQR (default PR).

sampling: The approach for sampling candidate subsets (default OWN).

samples: The number of candidates to sample

The algorithm uses SLF4J for logging and output, and can therefore work with any user-specified log4j properties file. In order to use this configuration, the log4j.properties file must be moved to the <SPARK>/conf folder, where <SPARK> is the path of the Spark installation being used. An example log4j.properties is provided with the following configuration.

- All output is redirected to different files in a folder called logs, which will be created at the time of the execution in the directory from which the Spark job was invoked.
- All output above the WARN level is written to logs/err.out
- All other output above the DEBUG level is written to logs/info.out
- All other output above the TRACE level is written to logs/trace.out

The results of the algorithm are written to logs/trace.out. The contents of this file can be described as follows:

- A list of the input parameters
- A list of the Spark configuration variables
- Information on the input matrix dimensions and other parameters, execution modes and partial execution times.
- The list of leverage scores (Probabilities)
- The size of matrix Q^T A (the number of rows equals the cardinality of the union of the candidate subsets)
- The total execution time in seconds (Algorithm time: x).
- The selected feature subset.
3.4.3. Example

A toy data set is available in example/toy_data. It consists of a 10x10 matrix with three sets of numerically dependent columns (linear combinations with additive Gaussian noise). Below is an example of an execution keeping 3 columns, computing the SVD, standardizing the data and requesting 4 partitions. Please make sure that the example log4j.properties files was copied to the <SPARK>/conf folder, where <SPARK> is the path of the Spark installation being used. Otherwise, the output of the execution might be lost.

```
spark-submit target/pics-1.0.jar --file="example/toy_data" --k=3 --svd=TRUE --mp=4
```

The result of running this example is shown in Figure 6, which shows the contents of the file logs/trace.out (using the bundled log4j.properties file).

![Figure 6 The output after running PICS on a toy data set](image)

The input arguments and the Spark configuration are shown first. Then, some information such as the size of the input matrix, an estimation of its numerical rank and the resulting column sampling probabilities is given. The resulting subset is indicated by the line “Chosen subset:”. Next, the attained residual norm with this subset and the one obtained using the singular value decomposition are shown.
3.4.4. Development status

Research is currently underway to improve PICS in various regards. First, we intend to make the computation of the Frobenius norm more efficient in order to allow faster evaluation of larger numbers of candidate subsets. Second, we plan to add regularization terms to improve the generalization ability of the algorithm.

3.5. IterFS: Deterministic feature selection for unsupervised learning (UPM)

IterFS is a deterministic algorithm that provides an approximate solution to the Column Subset Selection Problem. Its purpose is similar to that of PICS (see above), but it is suited for different scenarios. IterFS is deterministic given an initial subset, and can provide results generally faster than PICS. However, it yields only one model. Therefore, PICS might be better suited for scenarios where different models might be interesting for exploratory data analysis, while IterFS should be used when the data must be processed quickly.

IterFS has an interesting property that allows it to process matrices of any number of rows. Its output is invariant to orthogonal transformations, which in practice means the following: if we need to process a matrix of M rows and N columns, we can transform it into a square matrix of N rows and N columns and obtain the same result. All that is needed to benefit from this fact is to run a decomposition that yields an orthogonal subspace of our data. The singular value decomposition, which is efficiently implemented on Spark, will suffice.

3.5.1. Download and Installation

1. Clone the IterFS repository

Use the following command to clone the IterFS repository.

```
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./WP3-CSE/iterfs
```

2. Dependencies

IterFS requires the following software to run:

- Python 2.7
- Numpy: [http://www.numpy.org](http://www.numpy.org)
  - Numpy is the de facto standard library for scientific operations in Python.
- The OLA library (binary bundled with IterFS)
  - OLA is a library developed by the UPM for specialized numerical algebra computations.
3.5.2. Deployment

The algorithm does not require installation. It can be run as specified in the synopsis below.

```
python iterfs.py -f|--file=<file> <k> [-s|--std, -d|--dump]
```

- `-f|--file`: The path to the input data matrix file (in whitespace-separated values format)
- `-k`: The number of columns to choose
- `-s|--std`: If present, the data will be standardized to zero mean and unit variance on entry
- `-d|--dump`: If present, the standardized data will be written to a file

Output:

A set of column indices indicating the ones that were chosen

IterFS writes its results to standard output in the manner described in the example shown below.

3.5.3. Example

A toy data set is available in example/toy_data. It consists of a 10x10 matrix with three sets of numerically dependent columns (linear combinations with additive Gaussian noise). Below is an example of an execution keeping 3 columns.

```
python iterfs.py -k 3 -f example/toy_data
```

The resulting output is as shown in Figure 7.

![Output of IterFS on toy data set](image)

Figure 7 The output after running IterFS on a toy data set
The algorithm starts from an initial subset and iterates replacing its columns so as to maximize its approximating power. The result of each iteration is written to standard output. When the algorithm converges, the resulting subset as well as the attained residual norm is shown.

3.5.4. **Development status**

The UPM team is currently working on the parallelization of IterFS to extend its applicability to domains of extremely high dimensionality. Regularized variants are also being considered, as well as techniques to decrease its complexity with respect to the number of chosen columns. Methods to find adequate choices for the number of columns to choose are also being explored.

In addition to these research challenges, the UPM team plans to integrate IterFS with Spark to facilitate its usage on large data sets.

3.6. **PSCEG: an unbiased parallel subspace clustering algorithm using exact grids (UPM)**

PSCEG is a new parallel grid-based subspace clustering algorithm designed to achieve better scalability, accuracy and efficiency on massive high-dimensional datasets than existing algorithms. The quality of grid-based subspace clustering results is highly dependent on the grid size and positioning, and many existing methods use sensitive global density thresholds that are difficult to set a priori and will cause a bias to a certain dimensionality. Our algorithm PSCEG can capture the positions of dense units and proper size of the grid based on the data distribution of each dimension without the need to specify its size a priori. To avoid the effect of dimensionality bias, PSCEG conducts subspace clustering with different granularities by defining and using a novel unbiased density estimator that is adaptive to dimensionality.

The parallelization of PSCEG is based on Apache Spark. This increases the scalability, making the structured input classifiers usable in big data scenarios and, more specifically, for data-intensive 5G applications.

In the 5G context, datasets are expected to represent very complex phenomena represented in hundreds or thousands of features. Therefore, clusters associated with different natural processes may exist in different subspaces of the full feature space. Subspace clustering techniques were proposed to discover clusters that exist in relevant feature subsets.

Within the CogNet project, we intend to use PSCEG for different use cases and scenarios.

- Collaborative Resource Management: even though this use case has a clear goal in mind (improving our ability to classify network traffic) it also has an exploratory component. The data currently being collected in the Mouseworld environment for this use case is complex and high dimensional. Hence, PSCEG can be helpful in analysing it and to reveal interesting behaviour.
3.6.1. Download and Installation

This project is based on PySpark, the Python API of Apache Spark. Two libraries are needed to successfully run the program: numpy and sklearn. Follow the instructions below to install the required tool.

1. **Clone PSCEG from the repository**

Use the following command to clone PSCEG from the repository.

```
$ git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./WP3-CSE/PSCEG
```

2. **Dependencies**

IterFS requires the following software to run:

- Python 2.7
- Numpy: [http://www.numpy.org](http://www.numpy.org)
  
  - Numpy is the de facto standard library for scientific operations in Python.
  
  - Scikit-learn provides implementations of a large variety of machine learning methods

3.6.2. Deployment

The command to run the algorithm using PySpark is as follows:

```
$ spark-submit psceg.py <InputFilePath> <OutputDirectoryPath> <NumPartitions> <alpha> <theta>
```

Input filename: name of the input file that contains the input data. Each line of the preprocessed input file should be a list of numeric float values in the range [0,100] representing a data point. The values must be separated by whitespace characters. The input file should be preprocessed first to make sure that the values of each column are rescaled to the range [0,100].

Output filename: name of the output file where the subspace clustering results will be stored.

Number of partitions: the minimum number of partitions for the input file, as specified by Spark.

Alpha: a parameter called cluster dominance factor that estimates the density of the expected clusters.
Theta: a parameter that regulates epsilon and minPoints for DBSCAN.

The results (i.e., the set of subspace clusters) can be found in the specified output directory. Each paragraph of the output file contains one subspace cluster. Each subspace cluster is represented by a unique identifier, its size, and the set of indexes of the data objects it contains. Each paragraph in the output file is in the form:

(\text{clusterIndex}, \text{subspace}[\text{dim1}, \text{dim2}, \ldots], \text{sizeOfCluster}, \text{setOfDataObjects}[o_1, o_2, \ldots])

For instance

\[(1, ([0, 1, 2, 3], 593, [0, 1, 2, 3, 4, 5, 6, 7, 8, \ldots 597, 598, 599]))\]

represents a cluster in the subspace [0, 1, 2, 3] that contains 593 data objects.

3.6.3. Example

An example dataset is bundled with PSCEG. Use the following command to try the algorithm on these data.

\[\text{spark-submit psceg.py example.txt outputFile 40 1.5 0.02}\]

The output is shown in Figure 8.

![Figure 8 The result of running PSCEG on a toy data set.](image)

3.6.4. Development status

PSCEG is provided in its final version. We are currently working on more efficient algorithms for subspace clustering which are not strongly dependent on the input parameters.

3.7. Funco: Model-based co-clustering for functional data (Orange)

In the 5G context, we are focusing on Network automatic self-healing and more specifically in the automation of troubleshooting and optimization. Therefore, we aim to
analyse the behaviour of the Key performance indicators of a geographical zone in order to help the technical support team with their daily decision-making.

In this module, we present a model-based co-clustering algorithm. Our algorithm is applied to functional data i.e. we treat information on curves or functions. We aim to explore the hidden double-structure in the dataset.

We start with a pre-processing step: for each curve, we apply a smoothing technique. Next, we apply a univariate functional principal component analysis on all the smoothed curves. Later, on the resulted principal components, we apply a co-clustering. Our co-clustering is a variety of Latent bloc modelling by assuming that each co-cluster is defined by its Gaussian distribution. The model is parametric and the estimation is done by a SEM-Gibbs algorithm.

Funco is implemented with R as programming language and we refer the reader to [8] for more details. The algorithm can be used as a co-clustering for functional data. It also can be used as a clustering technique of multivariate and (also univariate) functional data (when the number of column clusters is equal to one), and also it can be updated for co-classification purposes.

Within the CogNet project, we will use our model for the self-healing problematic in the WP5. We will be able to discern normal and abnormal behaviours of the KPIs. This clustering is useful in the first phase of the troubleshooting which is the problem identification. It will allow us to predict some problems that may occur in the future.

Within this deliverable, we provide the code for the deployment of the model and in what follows, we provide the technical documentation.

### 3.7.1. Download and Installation

This module requires R or RStudio to be installed. If you do not have R installed on your system, please refer to the following page for instructions on how to install R: https://cran.r-project.org/doc/manuals/R-admin.html

The module needs some R packages to be installed as well: `mvtnorm`, `fda`, `mclust`, `Hmisc`, `combinat`. If not installed, you can follow the previous link for instructions on how to install an R package.

Use the following command to clone the functional co-clustering model from the repository.

```
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./WP3-CSE/Funco
```

### 3.7.2. Deployment

#### The input file format

The model deals with a matrix of functional data as described in [8]. Figure 9 illustrates an example of 6 observations with 8 functional attributes.
While creating an extract of real data, the values issued from the probes are discretised; therefore the dataset is described by a 3 dimensional array $n \times d \times p$

- $n$: The number of observations (Lines)
- $d$: the number of discretised values in the curves
- $p$: the number of attributes (columns)

However, during the extract of real data, most of the applications use a 2 dimensional format like an excel file or a csv file, thus, in the input file, two dimensions will be merged.

- Either, $(n-d)$ are merged in lines and $p$ is put in columns
- Or $(n-p)$ are merged in lines and $d$ is put in columns

We apply a pre-processing step to create the 3 dimensional dataset. Then we apply a smoothing step to regain the functional behaviour from the discretised curves. Next, we apply a Functional principal components analysis to reduce dimensions. After pre-processing, the dataset is a 3 dimensional array of size $n \times h \times p$

- $n$: the number of observations (Lines)
- $h$: the number of principal components
- $p$: the number of attributes (columns)

**The model parameters**

The most important function is “Functional_coclustering.R”. It takes into parameters the pre-processed dataset and other parameters that can be changed by the user.

The usage of the function is as follows:

```r
Functional_coclustering(DS, kr_estim, kc_estim, nb_SEM, nb_simul_burn)
```

The arguments are:

- **DS**: the pre-processed dataset of 3 dimensions, just described above (n*h*p)
- **kr_estim**: the number of line clusters
- **kc_estim**: the number of column clusters
• **nb_SEM**: the number of iterations of the SEM-Gibbs algorithm

• **nb_simul_burn**: the number of the burn-in period of the SEM-Gibbs algorithm

If \( Kc_{estim} = 1 \) and \( p > 1 \), we are in a case of multivariate clustering and it will be treated by “Functional_clustering.R”.

If \( kc_{estim} = 1 \) and \( p = 1 \), it is a univariate clustering and it will be treated by “Functional_clustering_univariate.R”.

**Execution**

In order to execute end-to-end functional co-clustering, including all the preliminary steps, we provide a script, “Training_model.R”. In order to run this script, go to the project folder:

$ cd dir/to/project/Funco

The command for training the model is:

R CMD BATCH [options] Training_model.R

• **[options]**: Options that can be used with the Batch execution of R as mentioned in [https://stat.ethz.ch/R-manual/R-devel/library/utils/html/BATCH.html](https://stat.ethz.ch/R-manual/R-devel/library/utils/html/BATCH.html)

• **Training_model.R** is the function in which the input file is read and preprocessed and in which the model parameters are fixed and the coclustering model is called.

The output of this command is:

• **Training_model.Rout** -> a file containing the training results in term of time execution and the description of the model

• **model.RData** -> the stored learned model

• Some plots to test the convergency of SEM-Gibbs algorithm

In this **Training_model.R**, we start by randomly generating a training set. The user can change the values of the variables \( n, d, p \) (described in subsection 3.7.2), \( kr_{real} \) and \( kc_{real} \) which are the real number of line and column clusters, respectively.

The generated file will be saved in the “training_file.csv”. Each column is the discretized values of the curves (d values). Each line is an observation per attribute. So, each observation will be described by \( p \) attributes and therefore by \( p \) lines in the file. The total number of lines is therefore the number of observations multiplied by the number of attributes. The first line is a header line.

If the user has his/her own training file, one can ignore the randomly generation step and set the path to the specific training file. But, he/she will need to specify the values of:

• **zr_real**: a vector describing to which line cluster each observation belongs

• **zc_real**: a vector describing to which column cluster each attribute belongs
They are only used to compute the performance of the model by comparing the real co-clusters to the estimated ones.

In order to apply the pre-processing on the file, the user may change the values of the following parameters:

- **nbasis**: number of basis functions for the smoothing (nbasis must be smaller than \( d \))
- **nb_scores**: number of principal components (Nb_scores cannot be greater than nbasis)

After the pre-processing step, the Functional_coclustering function will be called and therefore, the user may change the parameters \( k_r_{\text{estim}} \), \( k_c_{\text{estim}} \), nb SEM and nb_simul_burn described above. All those parameters can be changed directly in the script **Training_model.R**.

**Example**

The toy dataset can be found in **training_file.csv**. It is a file with training data containing 50 observations, 20 attributes. Each curve is described by 30 discretised values. The columns are the discretised values. The lines are the observations per attribute.

Apply the following command to run the experiment:

```
R CMD BATCH Training_model.R
```

In this example, we set the parameters to:

```
n=50 # number of observations  
p=20  # number of attributes  
d=30 # number of discretised values of each curve (of each attribute and each observation)  
k_r_{\text{real}}=3  
k_c_{\text{real}}=3  
nbasis=10 # number of basis functions for the smoothing  
nb_scores=5 # number of principal components  
k_r_{\text{estim}}=3 # the number of line partitions to be estimated  
k_c_{\text{estim}}=3 # the number of column partitions to be estimated  
nb_{\text{SEM}}=20 # number of SEM-Gibbs iterations  
nb_{\text{simul_burn}}=15 # the burn-in period of SEM-Gibbs algorithm
```

Figure 10 demonstrates a console output of the functional co-clustering model applied on the training_file.csv. The tool reports the time execution of the model, the adjusted rand index for both the line clusters and the column clusters and the estimated parameters of each co-cluster. Figure 11 presents some plots of the parameters convergency test of the SEM-Gibbs algorithm.
3.7.3. Development status

Currently, we defined a model-based co-clustering for functional data. We implemented the code using R language. The implementation concerns the training model for co-clustering of functional data and the special cases of clustering for both multivariate and univariate functional data.

In the second prototype release, we will add the Spark support for our code. We will deliver an R package and a version of the model with better performance and a better optimisation of the code. The algorithm is sensitive to the initialization phase, we will propose better initializations.
3.8. ML4MQ : Mobile User Throughput Prediction (Orange)

ML4MQ is a module able to estimate the achievable throughput of a user thanks to features describing its device radio context and the performance of the radio access network. This module is a part of SLA enforcement scenario and belongs most particularly to the SLOs breaches identification at service level. If the predicted throughput is not sufficient to guarantee a good level of QoS for a certain kind of services, it can be seen as a SLA violation between the user and the mobile network operator.

More specifically, its output could be used by video content providers to estimate bandwidth before launching delivery. Currently, adaptation of content is made after the first exchange between client and distant servers, our module is able to give indications to contents providers before these first exchanges and thus allows content provider to deliver adapted contents before the beginning of the video.

A continuous throughput value is predicted thanks to a supervised learning. We used Spark environment and Python language to train and test this module based on the Random Forest algorithm. We provide the model, the python script and an anonymized dataset available under the Github CogNet repository to run the module on. The python script loads the model previously trained, then it loads the features into Spark to creates a RDD (resilient distributed dataset) and finally it uses the model with the features to predict the throughput. The result is stored outside Spark in a predefined folder (see 3.8.2 for more precise information on how to use the python script).

The proposed model was trained on the data coming from two distinct sources: the user’s radio context features coming from the smartphone and the radio access network performance KPIs coming from the NMS (Network Management System). In order to collect these two distinct datasets, we installed a specific mobile application on several smartphones. This application downloads periodically a file from a server and collects KPIs related to radio context. The rest of the data features have been provided by the NMS at each download.

In the CogNet context, these data could come from a VNF dedicated to NMS function, and the data coming from smartphone could be accessible thanks to MDT standard (3GPP TS 32.432, 3GPP TS 32.421).

In order to train our model, we used technologies mentioned above to capture the following data: Throughput, cell Band, RSRP, RSRQ, "Radio Access Network Performance KPIs". Using these data, we trained a model that predicts the throughput using 'cell Band', 'RSRP', 'RSRQ' and 'Cell Performance KPIs' as features

The Cell Performance KPIs are as follows: Call Setup success Rate, Average number of user, Setup Success Rate of the Radio Access Bearer, Average of Active user, Average downlink throughput, Number of Radio Resource Control Attempt, Average of User Equipment connected to Radio Resource Control, Maximum number of User Equipment connected to Radio Resource Control, Downlink Bloc Error Rate

All the observations have been produced during a file download when the smartphone was attached on the French Orange 4G mobile network.
We tested the model on a dataset of 653 observations, coming from the French Orange mobile network, achieving 0.83 in terms of coefficient of determination, $R^2$.

### Compute determination coefficient : $R^2$ squared (2)###

```python
R2_num = TargetAndPredictions.map(lambda (lab,pred):(lab,pred)*(lab-pred)).sum()
R2_denominator = TargetAndPredictions.map(lambda (lab,pred):(lab-pred)**2).filter(lambda (lab,pred):lab==TargetAndPredictions.count)

R2 = R2_num/R2_denominator

print "R2='1.0-\n  \"".R2/\n  \"=\g2\n  \"n\n  \"\n  R^2=1-\n  \"\n  \"=\n  \"\g2\n  \"\n  \"g2\n  \"\g2\n  \"g2\n  \"g2\n  \"g2\n  \"\n
R2 = 0.832374601993
```

![Figure 12: Results on validation dataset](image)

$R^2$ formula is depicted below, more $R^2$ is close to 1 better it is.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2}{\sum_{i=1}^{n}(\bar{Y} - Y_i)^2}$$

Where $n$ is the number of observation, $\hat{Y}_i$ is the predicted value of the i-th observation, $\bar{Y}$ is the average of the target value calculated on the test dataset and $Y_i$ is the target value of the i-th observation. The closer $R^2$ is to 1, the better it is. On the screenshot below you can see the result of our predictive model based on 800 Trees, with the value of $R^2$ and the result of the first 10 predictions. More detailed information on $R^2$ is available at [https://en.wikipedia.org/wiki/Coefficient_of_determination](https://en.wikipedia.org/wiki/Coefficient_of_determination).

### 3.8.1. Download and Installation

There are no specific hardware requirements to create, test or run the predictive model.

Python and Spark must be installed under Linux to run the module (we used python 2.7 and Spark 1.6) then the “py4j” package must be installed to use pyspark. If not, launch the following command:

```bash
sudo pip install py4j
```

Use the following command to clone the module from the github repository:

```bash
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./ML4MQ
```
The ML4MQ folder contains a pretrained model, a dataset example and a python file to be used to launch the model.

### 3.8.2. Deployment

To use the model under a Linux environment with Spark, type the following command:

```bash
spark-submit ml4mq.py input_data model_dir output_dir
```

Here, the arguments are as follows:
- `input_data` -> data awaited by the module for throughput prediction
- `model_dir` -> directory of the provided model
- `output_dir` -> directory to store the results

The input data format is as follows. It should be a simple text file with the comma-delimited columns without any header.

The first column of each line contains the user ID, and is followed by 12 features columns containing the following groups of features:

**User’s radio context features:**

1. cell Band : Frequency band of the cell
2. RSRP : Reference signal Received Power
3. RSRQ : Reference Signal Received Quality

**Radio Access Network performance features:**

4. cssr : Call Setup Success Rate
5. avg_cell_user : The average number of user in a cell
6. rab_ssr : Ratio between the success of RAB setup and the total number of attempt of RAB setup
7. avg_active_user : Average of active users in a cell
8. avg_down_throughput : The average of user’s downlink throughput in a cell
9. nb_rrc_attempt : The number of RRC attempt in a cell
10. avg_rrc_user : Average of user’s equipment connected to RRC in a cell
11. max_rrc_user : Max number of user’s equipment connected to RRC in a cell
12. down_bler : Block error rate of downlink flow in a cell
13. ...

Each radio access network performance feature is calculated by cell, over a certain period of time, while the user’s radio context features are measured at the time when the file download is performed. To obtain a consistent dataset composed of these two data sources, the collected radio access network performance features must have been computed on a period of time that include the file download. In a same way the two data sources must be linked by the cell identifier. Finally, observations with a handover that occurred during a file download have not been taken into account.

In our distribution, we provide a pretrained model, `./ML4MQ/model`, and a sample anonymized test file, `./ML4MQ/test`. In order to run the throughput prediction on our sample data using the pre-trained model, launch the following command:

```bash
spark-submit ml4mq.py example/input_data.csv model predictions
```
The output of the model is a simple CSV with only two columns as it is described below:

1. User_ID: Identifier of the user
2. predicted throughput: Value of the predicted throughput

![Figure 13 ML4MQ output](image)

### 3.8.3. Development status

These works demonstrate the possibility to predict user’s throughput thanks to radio and cell performance KPIs with a good reliability. But, most applications simply need a throughput class rather than a precise continuous value for throughput. Class prediction is planned in the second version of the module.

### 3.9. TCDC: Model Recommendation for Supervised Learning (IBM)

Nowadays, there are thousands of machine learning techniques available and hundreds of new ones may be published every year [14]. Meanwhile, there is no single model that will always outperform other models, which is known as the no free lunch theorem [15]. Therefore, choosing an optimum machine learning model according to the problem is a challenging task. In CogNet, the CSE has no prior knowledge on models suitable for given datasets and is agnostic to analytics tasks. As a result, models need to be identified on-the-fly. To address the challenge on model selection, we introduce a high-level approach called TCDC (train, compare, decide, and change) that shows how to select an optimum model for either classification or regression according to the dataset’s characteristics, which has been published in [3]. This approach will be deployed in the algorithm selection component of the CSE to facilitate analytics tasks in CogNet.

The TCDC is comprised of four main phases:

- **Train**: two models are mainly trained in this phase based on the target dataset. (a) The most flexible but least interpretable model available in the toolbox which will not change through the comparison, which is served as a reference in following analysis and (b) the simplest model available in the toolbox, i.e. the model which is the least computationally expensive, and is easy to interpret and to implement. We apply SVM as the reference model in our work and its
implementation, since it has been found to be very powerful across many problem domains and easy to use in practices.

- **Compare:** the predictive performance, and other related metrics of the two models trained from the previous step are compared in this phrase.
- **Decide:** the predictive performance of the simpler model is checked in this phrase to find out whether that is acceptable compared to the reference model. The simpler model will be chosen if it has an acceptable performance.
- **Change:** If the solution has not found the optimum model with acceptable performance compared to the reference model, this phase is triggered. In this phase, a more complicated model is selected and gone through the above phases again, which aim to find out the optimum model.

The TCDC solution has been implemented as a Proof of Concept (PoC) in R, and packaged in R functions that can be invoked by any R codes. It is delivered in D3.2 as a support piece for the development of CSE. The code is developed based on the caret package, which and its support packages should be installed and configured correctly before invoking the model selection functions.

### 3.9.1. Download and Installation

The R libraries associated with test data can be cloned by

```
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd ./WP3-CSE/ModelSelection
```

This work assumes that users have already explored the relationship between features and outcome variables, and hence, discovered whether a given target is linear or non-linear. Based on that, the model selection work consists of four functions:

1. **linearClassification/nonlinearClassification** -- it recommends a classification model that offers the highest or very close to the highest accuracy for the linear/non-linear related features.
2. **linearRegression/nonlinearRegression** -- it recommends a regression model that offers the lowest or very close to the lowest Root Mean Square Error (RMSE) for the linear/non-linear related features.

The four functions are packaged in four R files, which should be placed in the folder that can be invoked in R codes. Meanwhile, to ensure the smooth execution of these functions, the packages supporting the following models, should also be installed and configured correctly for the “caret”:

1. SVM, PLSDA, LDA, Logistic Regression, Nearest Shrunken Centroid, when dealing with linear classification
2. SVM, FDA, Naive Bayes, Random Forest, Stochastic Gradient Boosting, CART, C45, RDA, MDA, when dealing with non-linear classification
3. SVM, PLS, Elastic Net, PCR, OLR, Lasso, Ridge Regression, when dealing with linear regression
4. SVM, MARS, Model Tree, Random Forest, Stochastic Gradient Boosting, when dealing with non-linear regression

Note that if any of the packages above are missing, you need to follow the reminder information from R to install them.

3.9.2. Deployment

The four functions come with public available datasets (they are also available in the git folder) for validation and test purpose, including:

1. Iris (available in standard R) and bank note datasets to validate the classification;
2. Concrete compressive strength and airfoil self-noise datasets to test the regression.

Users can invoke the functions, like any other external functions in R, by

```
functionName<dget('r_function_name.R')
functionName (features, target, [trackingState])
```

The “r_function_name” can be linearClassification/nonlinearClassification or linearRegression/nonlinearRegression.

The parameters for the functions, include:

- features: the input feature of the data
- target: the target of classification problem
- trackingState: Boolean, when the value is ‘true’, test results from each checked algorithms will be displayed. Its default value is ‘false’.

The return of the functions includes either the algorithm that offers the highest or very close to the highest accuracy for classification, or the algorithm that offers the lowest or very close to the lowest Root-Mean-Square-Error (RMSE) for regression.

One example is given in Figure 14. It illustrates how the work suggests a model for linear regression problems based on Concrete Compressive Strength dataset. Compared with the outputs of all models considered in the work, which is listed in Figure 15, the TCDC solution suggests an optimal one.

A script that implements the example above is given in the Git repository with the libraries. This script, named “test.R”, illustrates how to invoke our tools to perform model selection for regression based on the given dataset. Before running the script, set

8 available from [https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength](https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength)
the path to the working directory in its line 5 (\texttt{setwd("<put here path to your working directory>")}). The example script can be run as follows:

Rscript test.R

When adjusting the script to your need, please note the features and target of the model should be clearly identified in the script, such as those given in the line 7 and 8 of Figure 14. Once your R script is completed, it can be invoked by command line through "Rscript nameOfFile.R".

3.9.3. SPARK support

The current version of model selection does not support Spark. The integration of this work with Spark is under study.

3.9.4. Development status

A prototype that validates the work proposed in \cite{3} has been fully implemented. It may be enhanced as the further investigation in the area by the IBM team based on the
requests from the project. Development work will adapt to the latest achievement on the theoretical research.

3.10. Distributed Application Performance Optimizer
(Vicomtech)

The Distributed Application Performance Optimizer modules refer to the Massive Multimedia Consumption CogNet scenario, as well as potentially to the scenarios related to Connected Cars and to the Follow the Sun scenario.

The general purpose of the modules that compose the Application Performance Optimizer is to serve — in the specific framework of those two scenarios as well as in any other that might need them in the course of the project in particular — as building blocks for the definition and setup of prediction and optimization components that can provide improvement suggestions to network operators about the structure of their networks.

The key activity foreseen in relation to this functionality for the rest of the project is the set-up of a real world experiment in which the configuration of a Service defined in terms of Network Virtualized Functions is automatically optimized based on the results of a regression / forecasting of metrics of relevance based in turn on a description of a current situation that is summarized by describing the natural group it belongs to. All the modules in the Application Performance Optimizer will in principle be involved in this activity.

The Application Performance Optimizer is comprised of three modules, a Clustering module, a Regression module and an Optimization one:

- The Clustering module can be used to understand the status of a network which is being used for delivering Massive Multimedia data flows in conditions of varying traffic
- The Regression module can be used for forecasting KPIs of a Massive Multimedia delivery service
- The Optimization module can be used to help human operators enforce SLOs while keeping under control the operating costs.

The overall diagram of the system is depicted in Figure 16.

This would be roughly the processing flow of the implemented system, where several sub-modules are shown. They are briefly presented below:

- **Unsupervised Classifier — the “Clustering” module:** A module to continuously detect the status of the network load. For that aim, it ingests data from the database where metrics of the network have been stored and updates the classification model each time.
- **Regressor — the “Regression” module:** It takes in the output of the classifier plus the rest of the metrics of the system and predicts some key features of the model to be fed to the optimizer.
- **Optimizer — the “Optimization” module:** Takes into account the output of the machine-learning modules and simulates the performance of the all possible and permitted network configurations. It then passes the optimum arrangement
for the minimum cost, service time and other performance measures in order to comply with the SLA which the operator has to comply with.

- **Network**: Set of network elements such as servers, routers or switches that form the network infrastructure.
- **Metrics Database**: Monitoring element which measures and stores some key parameters related to the performance of the network.

### Figure 16 System block diagram

Classifier, Regressor and Optimizer modules, respectively implementations for the Clustering, Regression and Optimization methods, are described in Sections 3.10.1 (*streamCluster*), 3.10.2 (*Spark-streaming-linear-regression*) and 3.10.3 (*SAOptimizer*), respectively.

#### 3.10.1. *streamCluster*: Online classification module based on the Spark platform

This section describes the prototype of the real-time classification algorithm and the module that encloses it.

It has been based on the MLlib external toolkit, specifically on its KMeans classification algorithm (https://spark.apache.org/docs/latest/mllib-linear-methods.html).

The Clustering method can be used in the frame of WP4 and WP5 to understand the status of a network which is being used for delivering data flows in conditions of varying
traffic in terms of SLOs (WP4) and potentially security (WP5). It has been designed in order to address the Massive Multimedia content delivery scenario in particular.

Regarding the online classification module, there are several elements that conform it.

Figure 17 Classification module, block diagram

1. **Data Source**: Some kind of data structure or file where all the necessary data are stored in a standardized manner. By standardized it is meant that the agent that stores this data knows the syntax of the structure it has to comply with to be recognized by the data stream generator (server), which will be fed by this source.

2. **Data-Stream Server**: It is a module, which ingests the data that have been structured in the Data Source and serves it in a continuous flow to the Online Classifier. For that aim, it has to be listening in a TCP port defined for that purpose, and serve at each time the amount of data pre-defined by an user.

3. **Stream Controller**: Module to monitor and control the Data-Stream Server, which feeds the Online Classifier. It permits a user to ask for the current status of the server, as well as to modify its internal parameters such as amount of served data per batch/iteration, the data source, its cyclical behavior and others.
4. **Online Classifier:** This is the core module of the classification system. It is based on the Machine-Learning library of *Apache Spark* (MLlib) and it uses a K-Means classifier as the base algorithm ([https://spark.apache.org/docs/1.6.1/ml-clustering.html](https://spark.apache.org/docs/1.6.1/ml-clustering.html)). It defines two different input data streams, one for training and the other for testing/normal operation. Thus, two different instantiations of *Data-Stream Servers* are needed (apart from their respective *Stream-Controllers*).

Each time new data-points are received from the training data-stream, the internal model is updated in order to better fit the underlying system. At the same time, all the new data-points fed through the test data-stream are classified based on this criteria and a label is assigned to each of them.

5. **Out Data Stream:** Finally, the results produced in the *Online Classifier* module are forwarded to an output data stream, again implemented as a TCP socket, which provides the data to the next stage.

### 3.10.1.1 Download and Installation

Use the following command to clone the module from the repository.

```
$ git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
$ cd spark-stream-clustering
```

The operation of the classification module needs the Apache Spark engine to be installed. For that aim, the latest release can be downloaded (pre-built for a Hadoop distribution) from [http://spark.apache.org/downloads.html](http://spark.apache.org/downloads.html), but Apache Spark 1.6.1 prebuilt for Hadoop 2.6 has been used for the development of this module.

Once unpacked, the Spark folder must be placed in the working directory (spark-stream-clustering/).

Install pip and virtualenv:

```
$ apt-get install pip, virtualenv
```

Create a virtual environment and activate it:

```
$ virtualenv venv
$ source venv/bin/activate
```

Install the needed Python dependencies, using the provided requirements.txt file:

```
$ pip install -r requirements.txt
```

### 3.10.1.2 Deployment

In order to implement the depicted system, we have developed three different Python packages for this project: *streamCluster, socketStream* and *socketController*.

It has to be said that there is not a specific module to perform the training only. Training and classification are done simultaneously. This way, if new training data arrives the model is updated so that it can better classify the incoming stream.

As previously stated, the *Online Classifier* and its pythonic package *streamCluster* is based on the *pyspark.mllib* library, whereas both Data-Stream Server and *Stream*
Controller modules (and their respective socketStream and socketController packages) rely on the socket library from Python.

The Online Classifier module (see Figure 17) makes regular petitions through its training port (TCP socket) and the Training Data Server feeds new data in order to define the classification model.

To put to work the proposed online classification, the following set-up is needed. Note that we provide sample scripts which run toy examples in Section 3.10.1.2.1.

1) Open a Python interpreter
2) Define a SparkContext

```python
from pyspark import SparkContext
sc = SparkContext(appName='streamCluster')
```

3) Open a new terminal tab. Instantiate a training server.

**TAB 1. Instantiating the training server**

```python
from socketStream import TCPServer

TRAIN_PATH = ['path/to/source/']
TRAIN_FILENAME = ['filename.txt']
TRAIN_TCP_IP = ['host']
TRAIN_TCP_PORT = [port]
LINES_PER_BATCH = [n]
LOOP = [True/False]

srv = TCPServer(TRAIN_TCP_IP, TRAIN_TCP_PORT, LINES_PER_BATCH, LOOP)
srv.openSocket()
srv.loadFile(TRAIN_PATH, TRAIN_FILENAME)
srv.serve()
```

4) Open a new tab. Instantiate the training server controller.

**TAB 2. Instantiating the training server controller**

```python
from socketController import Controller

TRAIN_TCP_IP = ['host']
TRAIN_TCP_PORT = [port]

ctrl = Controller(host=TRAIN_TCP_IP, port=TRAIN_TCP_PORT)
while True: ctrl.displayOptions()
```

5) Open a new tab. Instantiate the test data stream server.

**TAB 3. Instantiating the test data stream server**

```python
from socketStream import TCPServer
```
6). Open a new tab. Instantiate the test data stream server’s controller:

**TAB 4. Instantiating the test data stream server’s controller**

```python
from socketController import Controller

TEST_TCP_IP = ['host']
TEST_TCP_PORT = [port]

ctrl = Controller(host=TEST_TCP_IP, port=TEST_TCP_PORT)
while True:
    ctrl.displayOptions()
```

7) Instantiate the classification module in new tab.

**TAB 5. Instantiating the classification model**

```python
from streamCluster import onlineKMeans

TRAIN_PORT = [port]
TEST_PORT = [port]
OUT_PORT = [port]
K = [cluster_k] # Number of clusters
INTERVAL = [interval_secs]

onlineClf = onlineKMeans(sc, TRAIN_PORT, TEST_PORT, OUT_PORT, K, INTERVAL)
onlineClf.defineModel()
```

8) Instantiate a Socket server at the output port

**TAB 6 Instantiating a Socket server at the output port**

```bash
netcat -l ['host'] [port]
```

9) Finally, return to the tab, where you instantiated the classification model (TAB 5). Note that the classification module does both training and classification at the same
time (i.e. it listens at the ports that feed the training and classification data at the same time).

**TAB 5** Running the on-line classification module

```python
try:
    onlineClf.start()
except KeyboardInterrupt:
    onlineClf.stop()
```

### 3.10.1.2.1 Sample scripts

We provide a number of sample scripts, which implement the above steps. Table 2 provides instructions on the list of commands to run. Note, that the scripts should be run in the order of their appearance. Here, the first column reports the command to run, the second column describes its functionality, and, finally, the “Tab ID” column reports to which of the steps in Section 3.10.1.2 it corresponds to. Additionally, in Figure 17, it is shown to which components of the classification module block diagram these scripts correspond to.

**Table 2** Code snippets for running prepared scripts

<table>
<thead>
<tr>
<th>Code</th>
<th>Functionality</th>
<th>TAB ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ python train_socket.py</td>
<td>Train Data Stream Server</td>
<td>TAB 1</td>
</tr>
<tr>
<td>$ python test_socket.py</td>
<td>Test Data Stream Server:  Serve test (classification) data stream</td>
<td>TAB 3</td>
</tr>
<tr>
<td>$ python train_socket_controller.py</td>
<td>Train Data Stream Server Controller: Monitor train data server</td>
<td>TAB 2</td>
</tr>
<tr>
<td>$ python test_socket_controller.py</td>
<td>Test Data Stream Server Controller: Monitor test data server</td>
<td>TAB 4</td>
</tr>
<tr>
<td>$ netcat –l 35007</td>
<td>Output Socket: Receive classification output</td>
<td>TAB 6</td>
</tr>
<tr>
<td>$ ./&lt;path to your spark distribution&gt;/bin/pyspark online_classifier.py</td>
<td>Online classifier: Classification module</td>
<td>TAB 5</td>
</tr>
</tbody>
</table>

Note that each code snippet has to be pasted to a separate terminal tab and that the virtual environment has to be activated beforehand (`$ source venv/bin/activate`), as described in Section 3.10.1.1, in order to have access to all the Python dependencies.
If run exactly as provided in the Code column, without setting any additional parameters, the scripts will deploy a system for a toy example with the values of all parameters predefined for a specific dataset supplied with this package, and specific port numbers.

Alternatively, the user can set the parameters for these scripts as follows:

**Train Data Stream Server**

```bash
$ python train_socket.py -path ['path/to/train_file'] -f ['train_filename'] -h ['train_host'] -p [train_port] -lpb [lines_per_batch] -loop [True/False]
```

The names of the parameters are self-explanatory.

Figure 18 illustrates the console output when running the above script with the default parameters.

![Figure 18 Set up and trace of training data server](image)

**Test Data Stream Server**

```bash
$ python test_socket.py -path ['path/to/test_file'] -f ['test_filename'] -h ['test_host'] -p [train_port] -lpb [lines_per_batch] -loop [True/False]
```

**Train Data Stream Server Controller**
Figure 19 illustrates the console output when running the above script with the default parameters.

```
$ python train_socket_controller.py -h ['train_host'] -p [train_port]
```

Test Data Stream Server Controller

```
$ python test_socket_controller.py -h ['test_host'] -p [test_port]
```

Output socket

```
$ netcat -l [out_port]
```

Figure 20 illustrates the same output of the above command when run on a toy example. Here, each line corresponds to the classification of a new input sample.

```
1: {'result': {'Label': 2.0}}
2: {'result': {'Label': 0.0}}
3: {'result': {'Label': 2.0}}
4: {'result': {'Label': 0.0}}
5: {'result': {'Label': 0.0}}
6: {'result': {'Label': 2.0}}
7: {'result': {'Label': 0.0}}
8: {'result': {'Label': 0.0}}
9: {'result': {'Label': 0.0}}
```
3.10.1.3 SPARK support

The provided code was tested with Apache Spark release 1.6.1.

3.10.1.4 Development status

We have defined a model-based online clustering module for multivariate datasets. The implementation has been based on Python. We have included some standalone scripts for model training and testing.

In the second prototype release, we will consider adding new online clustering engines (based on the DBSCAN algorithm), add some real datasets, as well as code optimisation for better overall performance.

3.10.2. Spark-streaming-linear-regression: Online linear regression with Spark’s Streaming API

Spark-streaming-linear-regression is a collection of sample scripts for performing online regression with Python and Spark’s Streaming API in the context of the “Regressor” module of the Distributed Application Performance optimizer (see Section 3.10) depicted as the ‘Regressor’ component in Figure 16 in Section 3.10).

The definition of Spark Streaming as defined by Spark’s Streaming programming guide is the following:

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark’s machine learning and graph processing algorithms on data streams.

A ‘DStream’ represents a continuous stream of data and is used to feed the data to the algorithm.

3.10.2.1 Download and Installation

Use the following command to clone the module from the repository.

\[
\text{git clone https://github.com/CogNet-5GPPP/WP3-CSE.git}
\]
\[
\text{cd ./WP3-CSE/spark-streaming-linear-regression}
\]

\[\text{10 https://spark.apache.org/docs/latest/streaming-programming-guide.html}\]
The operation of the online linear regression module needs the Apache Spark engine to be installed. For that aim, the latest release can be downloaded (pre-built for a Hadoop distribution) from their website\textsuperscript{11}.

Once unpacked, the Spark folder must be placed in the working directory.

Install pip and virtualenv:

\begin{verbatim}$ apt-get install pip, virtualenv$
\end{verbatim}

Create a virtual environment and activate it

\begin{verbatim}$ virtualenv venv$
$ source venv/bin/activate$
\end{verbatim}

Install the needed Python dependencies, using the provided requirements.txt file

\begin{verbatim}$ pip install -r requirements.txt$
\end{verbatim}

3.10.2.2 Deployment

According to Wikipedia’s definition, linear regression\textsuperscript{12} is an approach for modelling the relationship between a scalar dependent variable $y$ and one or more explanatory variables (or independent variables) denoted $X$. In offline linear regression, the model is fitted with a given set of $(X,y)$ tuples. In online linear regression, only one $(X,y)$ tuple is available at each time and the model is trained and updated as each new tuple becomes available. This section contributes sample code to perform the model training in two ways: using a built-in function and using a custom algorithm.

There are two options for performing streaming (online) linear regression with Spark’s Streaming API: using the built-in StreamingLinearRegressionWithSGD\textsuperscript{13} algorithm or implementing a custom algorithm. This document uses a running example to illustrate these options. The dataset used in the running example comes from a simulation of a communication network’s traffic, as explained next:

Networks with 3 different types of topologies (linear, star and toroidal) and different number (cardinality) of nodes (2, 4, 8, 16 and 32) are created ($3 \times 5 = 15$ different configurations in total). For each of these networks, data has been collected to characterize the performance of the GStreamer service in each configuration created. Moreover, to simulate network traffic, an Iperf was launched from the server to the last node on the network. In addition, Iperf allows measuring free bandwidth at each time (bandwidth not used by GStreamer). With this information, the network congestion can be analysed at all times. Once the network is created and the Iperf launched to create traffic, the GStreamer service is started. The GStreamer server is launched on the first network node and connected to a customer every 15 seconds. The service does not end

\textsuperscript{11} \url{http://spark.apache.org/downloads.html}
\textsuperscript{12} \url{https://en.wikipedia.org/wiki/Linear_regression}
\textsuperscript{13} \url{https://spark.apache.org/docs/latest/mllib-linear-methods.html#streaming-linear-regression}
until 15 seconds after the last client connects. Therefore, when the last client is connected, all clients in the network are connected to the server. Each GStreamer client collects metrics on latency, bit rate (bandwidth) and jitter of all packets received.

We provide two prototype scripts which perform online linear regression, both training and prediction, of the latency on the number of clients and path length for each topology. In both cases (StreamingLinearRegressionWithSGD and custom algorithm), the data is received through a socket and Spark converts it into a discrete stream (Dstream). Note that before deploying either of the examples, on first must set up a socket serving the data.

**Serving data through a socket**

In order to deploy the example, it is necessary to get data through a socket. This section shows how to read the data from a file and send it through a socket.

We generated sample data file created with the simulated communications: ./DATA/jons_ping_data.csv.

We provide a the socketstream.py script, which reads the data from the text file and serves it through a socket. The code to serve the data through a socket is executed in one terminal as follows:

```python
python socketstream.py
```

Currently, the script is predefined to read the data from the ./DATA/jons_ping_data.csv file. In order to set a new input file, please modify the following lines:

```python
# GLOBAL PARAMS
path = './DATA/' # path to source file
filename = 'jons_ping_data.csv'
stream_interval = 1 # sec(s)
```

**Example 1: mllib’s StreamingLinearRegressionWithSGD algorithm**

The pyspark.mllib.regression package provides an implementation of an online linear regression algorithm: StreamingLinearRegressionWithSGD.

Our script that performs regression with StreamingLinearRegressionWithSGD can be found in the repository with the name:

```python
Option1_StreamingLinearRegressionWithSGD_SOCKETS.py.
```

The training DStream received through the socket is split into three DStreams, one for each possible topology, and a StreamingLinearRegressionWithSGD model is trained for each stream. The user has to specify the hostname and port appropriately for the training and prediction streams. The output of the code is a stream of predictions for the test data stream.

To run this code in pyspark execute the following command in another terminal (different from the terminal running the code to serve the data through a socket) from the folder containing the code:
<path to the spark distribution>/bin/spark-submit
Option1_StreamingLinearRegressionWithSGD_SOCKETS.py 'localhost' 1432
'localhost' 1432

This same code is also provided as an ipython notebook, Option1_notebook.ipynb. To view notebooks in pyspark (Spark 1.6.1), execute the following command in the terminal:

IPYTHON_OPTS="notebook" ./spark/bin/pyspark.

**Example 2: custom algorithm**

We will use Walsh et al.'s online linear regression algorithm [16] to illustrate the implementation of a custom algorithm. The contributed code can be found in the repository in the Option2_walshetal_onlinelinreg_SOCKETS.py file.

In order to implement this algorithm, it is necessary to use the 'updateStateByKey' function (an explanation can be found in endgame.com[14], and example code can be obtained from Apache's github account[15]). Note that for stateful updates it is necessary to set a checkpoint[16].

The data in the DStream is used to train three models, one for each topology possible. As updateStateByKey is used in this case, the parameters of the models are added to the data DStream so that the algorithms have access to the previous state. The output of the code is a stream of predictions for the data stream.

To execute this code in pyspark execute the following command in the terminal:

./spark/bin/spark-submit Option2_walshetal_onlinelinreg_SOCKETS.py
'localhost' 1432

This same code is also provided as an ipython notebook, Option2_notebook.ipynb. To view notebooks in pyspark, execute the following command in the terminal:

IPYTHON_OPTS="notebook" ./spark/bin/pyspark.

### 3.10.2.3 SPARK support

The provided code was tested with Apache Spark release 1.6.1.

### 3.10.2.4 Development status

A prototype that works with toy data has been finished. We intend exploiting the developed and validated module in the frame of a large scale experiment dedicated to

---


the forecasting of KPIs for the enforcement of SLOs in a Massive Multimedia content delivery scenario.

3.10.3. **SAOptimizer: Optimisation algorithm for 5G networks**

As stated before, in the 5G context, we are focusing on automatic network management. This requires the optimization of some mathematical model representing the performance of the network. For solving this issue, the present task focuses on developing optimisation algorithms capable of improving the performance of the networks.

The SAOptimizer module is a Python library for the global optimisation of functions with or without constraints. The optimisation engine is based on the metaheuristic algorithm Simulated Annealing. Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space. It is often used when the search space is discrete. For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as gradient descent. These features of the simulated annealing algorithm make it a great choice for the Cognet project. In particular, the discrete capabilities of the SAOpimizer will help us in the optimisation of the topology of the network.

Apart from the simulated annealing algorithm, a steepest descent method has been coded. This method is executed after the simulated annealing and can be activated and deactivated by the user through the self.DO_FINAL_STEEPEST_DESCENT boolean variable.

The module is designed in Object Oriented terms. Public methods are documented according to the numpy docstring conventions (a PEP257 superset)

### 3.10.3.1 Download and Installation

Use the following command to clone the module from the repository.

```bash
git clone https://github.com/CogNet-5GPPP/WP3-CSE.git
cd optimization
```

SAOptimizer requires the following software to run:

- Python 2.7
- Numpy: [http://www.numpy.org](http://www.numpy.org) NumPy is the fundamental package for scientific computing with Python.
- NetworkX: [https://networkx.github.io/](https://networkx.github.io/) NetworkX is a software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
- Matplotlib: [http://matplotlib.org/](http://matplotlib.org/) is a plotting library which produces publication quality figures.
The SAOptimizer module can be imported with the following command:

```python
import SAOptimizer2
```

### 3.10.3.2 Deployment

We explain how to run a new optimisation function for the SAOptimizer module, on the example of the local sin optimisation problem.

First of all a new variable of class MinimizationProblem is created

```python
local_sin = MinimizationProblem()
```

Then the objective function is defined, in this case the sin function.

```python
local_sin.f = lambda xy: math.sin(xy[0])
```

The next step is to define the constraints of the problem, in this case the optimisation variable must be between $-\pi/2$ and $\pi/2$.

```python
local_sin.is_valid = (lambda x: -math.pi/2 < x[0] < math.pi/2)
```

The user must also set the starting point of the optimisation variable, zero in this case:

```python
local_sin.initial_best = [1.0]
```

Then a tuning parameter of the optimisation algorithm is set to 0.5:

```python
local_sin.start_radius = 0.5
```

Finally the mutation function for creating new candidates is defined:

```python
local_sin.sampler = SAOptimizer2.DefaultNDSampler(local_sin.is_valid)
```

In the `optimization` folder of our distribution, we provide the sample scripts which demonstrate the capabilities of the SAOptimizer library on the following cases:

- The sin function (`sin.py`): A minimisation problem has been defined for searching the minima of the sin function in the $(-\pi/2, \pi/2)$ domain.

- The Ackley function (`ackley.py`): The Ackley function is widely used for testing optimization algorithms. In its two-dimensional form, as shown in the plot above, it is characterized by a nearly flat outer region, and a large hole at the centre. The function poses a risk for optimization algorithms, particularly hillclimbing algorithms, to be trapped in one of its many local minima. More information can be found in [https://www.sfu.ca/~ssurjano/ackley.html](https://www.sfu.ca/~ssurjano/ackley.html)

- The Bukin function N. 6 (`bukin6.py`): The sixth Bukin function has many local minima, all of which lie in a ridge. More information can be found in [https://www.sfu.ca/~ssurjano/bukin6.html](https://www.sfu.ca/~ssurjano/bukin6.html)

- Network topology (`max_graph_connectivity.py`): A sample discrete problem for selecting the best network topology (adjacency matrix) for maximising the connectivity metric of the network.

The above benchmark functions were tested for checking the performance of the SAOptimizer. The following list reports the achieved results.
• **Minimisation of the sin function**

The function has a local minima in -\(\pi/2\) equal to -1. The initial point is equal to 1 by default.

Run the example as follows:

```
python sin.py
```

The optimisation algorithm is able to converge successfully to the desired solution as per Figure 21 and Figure 22, which illustrate the output of the above script.

• **Minimisation of the Ackley function**

The Ackley function presents a global minima at \(f(0,0) = 0\). The feasible region has been restricted by default to the -10 \(\leq\) \(x, y\) \(\leq\) 10 domain and the initial point is set to \(x = y = 10\).

Run the provided sample as follows:

```
python ackley.py
```

The SAOptimiser module reaches the correct solution avoiding the multiple local minima of the search space as per Figure 23.

![Figure 21 SAOptimiser convergence at runtime](image-url)
Figure 22: Minimization of the sin function

Figure 23: Minimization of the Ackley function
• **Minimisation of the Bukin function N. 6**

The Bukin function N. 6 presents a global minima at \( f(-10,1) = 0 \). The feasible region has been restricted by default to the \(-15 \leq x \leq -5\) and \(-3 \leq y \leq -3\) domain and the initial point is set to \( x = -10 \ y = 0 \).

Run the sample script as follows:

```
python bukin6.py
```

The SAOptimizer module gets stuck in a local minima \( f(-9.47,0.90) = 0.02 \). This can be considered normal due to the complexity of the Bukin N. 6 function.

An example run is described by the Figure 24.

• **Maximisation of the network connectivity**

The optimum network layout for maximising its connectivity is the complete graph, a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. In this example a 10 node network is used with an initial circular ladder configuration. Launch this example, using the command below:

```
python max_graph_connectivity.py
```

The optimizer converges successfully to the complete graph solution, as per Figure 25.

The next example shows the usefulness of the SAOptimizer modules in the context of the CogNet project. The CogNet project aims at designing a self-managing network that can monitor the environment, predict changes, both in the network performance and in the user demand and propose adequate network management solutions. This example illustrates how the number of edges in a topology are minimised while meeting a redundancy constrain in order to assure the stability of the network. In particular, the topology of a twenty node network will be optimised while requiring that it must be biconnected (meaning that if any vertex were to be removed, the graph will remain connected). As a result, even if one of the nodes fails, the network will remain connected. Launch the example as follows:

```
python min_edge_biconnected.py
```

Figure 26 and Figure 27 show the evolution of the optimal result.

As expected, the optimal biconnected network topology that minimises the number of edges is the circular. This shows the capability of the optimizer for improving the network performance.

3.10.3.3 **SPARK support**

The current version of the SAOptimizer does not support SPARK.

3.10.3.4 **Development status**

Currently we are working on the parallelization of the SAOptimizer. In addition, the integration of the SAOptimizer library with Spark is under study.
Figure 24 Minimization of the Bukin function N. 6

Figure 25 Maximization of the network connectivity
Figure 26 Log of the evolution of the optimal result

Figure 27 Evolution of the optimal result
3.11. Summary

In this section, we provided documentation for the distinct components of CogNet Smart Engine, the main analytical module of the CogNet project. All the modules are open-source and are available in a GitHub repository. They support (or will support in the next release) the same Big Data processing framework, Apache Spark.

The CSE components, described in this section, address various machine learning needs, such as feature selection, supervised and unsupervised classification and regression for various kinds of data representations. For each component, we describe how to install it, train a new model (if applicable), deploy it, and run a toy example. Each component’s description subsection is self-contained, so that its prospective users will need to read only it, and promptly pass to using this tool.

The components described here are the building blocks, upon which Work Package 4 and Work Package 5 will design their advanced solutions for the CogNet scenarios.
4. Conclusions

CSE is the main analytical component in the overall architecture of the CogNet project. Its purpose is the efficient analysis of the massive amounts of data reflecting information about the current network status. Its analytical results will be used when defining actions for the further network management.

CogNet project incorporates the main tasks to be solved when managing 5G networks into a number of scenarios. In Deliverable 3.1 [1] we have presented machine learning techniques necessary for addressing the tasks these scenarios and described their theoretical background. This deliverable is the first software release of the implementations of these components, available at the project’s GitHub, https://github.com/CogNet-5GPPP/WP3-CSE. The current implementation can be used as is, and we plan to further refine it when developing solutions to the CogNet scenarios in Work Package 4 and Work Package 5.

The release contains some new components originally not foreseen in the previous deliverable, which further expand the CSE’s capabilities. For example, one of them, TCDC (Section 3.9), enables the choice of the machine learning model, which is the best for a given task, not only in the terms of performance, but also in terms of easiness of implementation and interpretation as well as computational complexity. We list the new additional modules in Section 1 along with the references to the material that describes their theoretical background.

In the future stages of the project we will continue to refine the released software and to add Spark support to the components, which do not support it at the moment. While, currently, most models have been tested on the toy datasets, our next step is to apply them to the large-scale datasets employed in the Work Packages 4, 5, 6. We will perform both quantitative and qualitative analyses of their performance on the above data. Finally, as planned in DoW, we will experiment with creating ensemble models on top of the components, presented in the current release, and doing transfer learning and domain adaptation. We will present the new components along with the refined versions of the software components from the current release in the next CogNet engineering release, Deliverable 3.3.
References


[12] Anders Bjorkelund and Jonas Kuhn, "Learning structured perceptrons for coreference resolution with latent antecedents and non-local features.," in 52nd Annual Meeting of the Association for Computational Linguistics., vol. 1, Baltimore,
Maryland, 2014, pp. 47-57.


