
<table>
<thead>
<tr>
<th>Document Number</th>
<th>D4.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP 4</td>
</tr>
<tr>
<td>Deliverable Type</td>
<td>Other</td>
</tr>
<tr>
<td>Date of Delivery</td>
<td>M18</td>
</tr>
<tr>
<td>Responsible Unit</td>
<td>UPM</td>
</tr>
<tr>
<td>Contributors</td>
<td>Haytham Assem, Lei Xu, Teodora Sandra Buda (IBM), Udi Margolin, Elisha Rosensweig, Itai Segall (NOKIA), Imen Grida Ben Yahia, Jaafar Bendriss (ORANGE), Diego Lopez, Antonio Pastor (TID), Alessandro Moschitti, Raffaella Bernardi, Kateryna Tymoshenko, Azad Abad (UNITN), Alberto Mozo, Bruno Ordozgoiti, Jose Ramon Sanchez Couso, Jose Luis Lopez Presa (UPM), Ángel Martín, Marco Quartulli, Gorka Velez (VICOMTECH), Robert Mullins, Michael Barros (WIT)</td>
</tr>
<tr>
<td><strong>Reviewers</strong></td>
<td>Martin Tolan (WIT), Marius Corici (Fraunhofer) and Domenico Gallico (IRT)</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>Machine learning, NFV, network management, 5G.</td>
</tr>
<tr>
<td><strong>Dissemination level</strong></td>
<td>PU</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>10/10/2016</td>
<td>Working</td>
<td>Bruno Ordozgoiti (UPM)</td>
<td>Document template</td>
</tr>
<tr>
<td>0.2</td>
<td>07/12/2016</td>
<td>Working</td>
<td>NOKIA, UPM, UNITN, TID, Vicomtech, WIT</td>
<td>Contributions</td>
</tr>
<tr>
<td>0.3</td>
<td>09/12/2016</td>
<td>Working</td>
<td>Bruno Ordozgoiti (UPM)</td>
<td>Compilation</td>
</tr>
<tr>
<td>0.4</td>
<td>12/12/2016</td>
<td>Working</td>
<td>ORANGE</td>
<td>Contributions</td>
</tr>
<tr>
<td>0.5</td>
<td>19/12/2016</td>
<td>Working</td>
<td>Martin Tolan (TSSG), Marius Corici (FRAUN), Domenico Gallico (IRT)</td>
<td>Reviewed</td>
</tr>
<tr>
<td>0.6</td>
<td>21/12/2016</td>
<td>Draft</td>
<td>Bruno Ordozgoiti (UPM)</td>
<td>Compiled reviews and new contributions</td>
</tr>
<tr>
<td>0.9</td>
<td>24/12/2016</td>
<td>Draft</td>
<td>Bruno Ordozgoiti (UPM)</td>
<td>Final editions</td>
</tr>
<tr>
<td>1.0</td>
<td>30/12/2016</td>
<td>Finished</td>
<td>Alberto Mozo (UPM)</td>
<td>Final version</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable (D4.2) 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 their respective documentation. As opposed to the algorithms presented in deliverable D3.2, the components introduced here were developed within the considered scenarios, and intend to bridge the gap between machine learning methods and problems arising in automated 5G network management.

The scenarios that these components are being developed for are thoroughly described in D4.1 [1]. In particular, the components described here serve the following purposes:

- Time-series data pre-processing
- Automated model selection
- Text data classification for demand prediction
- Terahertz-band communication model enhancement

As the components reported in D3.2, the ones presented here are expected to scale to enormous data sets. Therefore, all the CSE components are expected to add Spark support during the second year of the project.
Table of Contents

1. Introduction ........................................................................................................................................... 7

2. CSE components ...................................................................................................................................... 10
   2.1. RelationalTextRanking: supervised pairwise Tweet classification ................................................. 10
       2.1.1. Overview .................................................................................................................................. 10
       2.1.2. Download, build and Installation ......................................................................................... 14
       2.1.3. Training .................................................................................................................................. 16
       2.1.4. Deployment ............................................................................................................................. 18
       2.1.5. Development status ................................................................................................................. 21
   2.2. TSDP: Time Series Data Pre-processing ........................................................................................ 22
       2.2.1. Overview .................................................................................................................................. 22
       2.2.2. Download, build and installation .............................................................................................. 22
       2.2.3. Deployment ................................................................................................................................ 23
       2.2.4. Preprocessing of the time series data ..................................................................................... 23
       2.2.5. Machine learning for time series ............................................................................................ 26
       2.2.6. Example .................................................................................................................................. 28
       2.2.7. Development status ................................................................................................................ 29
   2.3. Data pre-processing for demand prediction .................................................................................... 30
       2.3.1. Overview .................................................................................................................................. 30
       2.3.2. Download, build and Installation .............................................................................................. 30
       2.3.3. Deployment ................................................................................................................................ 31
       2.3.4. Development status ................................................................................................................ 33
   2.4. MoSeS (Model Selection Service) .................................................................................................... 34
       2.4.1. Overview .................................................................................................................................. 34
       2.4.2. Download, build and installation .............................................................................................. 34
       2.4.3. Deployment ................................................................................................................................ 35
       2.4.4. Example .................................................................................................................................. 36
       2.4.5. Development status ................................................................................................................ 37
   2.5. Connected Cars .................................................................................................................................. 38
       2.5.1. Overview .................................................................................................................................. 38
       2.5.2. Download, build and Installation .............................................................................................. 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.3.</td>
<td>Deployment</td>
<td>40</td>
</tr>
<tr>
<td>2.5.4.</td>
<td>Example</td>
<td>40</td>
</tr>
<tr>
<td>2.5.5.</td>
<td>Development status</td>
<td>42</td>
</tr>
<tr>
<td>3.</td>
<td>Conclusions</td>
<td>43</td>
</tr>
</tbody>
</table>
1. Introduction

There is a wide gap between the research and development of machine learning algorithms, models and theory, and their application to solve problems arising in the real scenarios. The main reason behind this is the fact that machine learning tools are in general heavily statistical in nature. Therefore, the performance of any machine learning algorithm is at most as good as the employed data. In addition, every machine learning model is built upon a set of assumptions, either explicit or implicit, about the behaviour of the analysed phenomenon. Thus, algorithms that perform well on a specific scenario may perform poorly when deployed to a different scenario.

A key element for the successful application of machine learning methods is the choice of an adequate model. A celebrated theoretical result known as the ”No free lunch” theorem [2] establishes that all algorithms are expected to perform equally well (or badly) when averaged uniformly over all possible problems. This implies that there is no reason to select one algorithm or model over another. It is therefore desirable to implement methods that enable the execution of this task in an automated fashion, in a way that it is integrated in an automated management pipeline.

The preparation of the available data sets for their consumption by machine learning algorithms is also essential to their correct performance. Variables often come in different scales due to the magnitudes of the phenomena they represent, and can suffer from missing values, e.g. due to faulty probing mechanisms. Therefore, mechanisms must be put in place to cope with these problems. Another approach to enhancing the data so that the chosen algorithm can be employed and its performance optimized is the transformation of the input into an adequate representation. This is especially true when the data is highly structured, which is the case, for instance, of images and time series. The latter are ubiquitous in network management environments, so tools that enable manipulation of time series can be extremely helpful for successfully applying machine learning methods in this domain. Even though current trends in machine learning advocate for the automatic learning of adequate representations, domain-expert knowledge can still be successfully exploited for this purpose in many practical applications.

In summary, effective use of machine learning to solve real-world problems always requires that a series of tasks are conducted to get the most out of both the data and the employed algorithms, such as data preprocessing activities, exploratory analysis and systematic model selection. This document describes a series of components that have been developed as enablers for the different scenarios being tackled by CogNet. Some of the components are specifically focused on the scenario they were developed for, while others can be exploited in a variety of environments.
Table 1 List of components described in this deliverable

<table>
<thead>
<tr>
<th>Component name</th>
<th>Scenario(s)</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RelationalTextRanking Java pipeline</td>
<td>Large-Scale events</td>
<td>Java</td>
<td>Conversion of raw text to structured representations</td>
</tr>
<tr>
<td>Pairwise tweet classification model</td>
<td>Large-Scale events</td>
<td>Java</td>
<td>Classification of tweet pairs as referring to the same event</td>
</tr>
<tr>
<td>TSDP</td>
<td>Follow the sun</td>
<td>Python</td>
<td>Time series data pre-processing</td>
</tr>
<tr>
<td>Data pre-processing for demand prediction</td>
<td>Urban Mobility Awareness</td>
<td>Python</td>
<td>Data pre-processing</td>
</tr>
<tr>
<td>MoSeS</td>
<td>Follow the sun Massive Multimedia Content Consumption</td>
<td>Python</td>
<td>Automated model selection integrated in infrastructure management pipeline</td>
</tr>
<tr>
<td>Connected Cars</td>
<td>Connected Cars</td>
<td>Python</td>
<td>Terahertz-band communication model enhancement</td>
</tr>
</tbody>
</table>

As the software presented in D3.2 [3], the components described here are intended for deployment in the CogNet architecture, specifically in the CogNet Smart Engine (CSE). The CSE adopts a Monitoring, Analysis, Planning and Execution (MAPE) Loop. The components offered here are designed so that their integration with other components is easy, serving as building blocks for enabling a smart management architecture.
The figure was originally presented in the Year 1 report of Work Package 2
2. CSE components

2.1. RelationalTextRanking: supervised pairwise Tweet classification

2.1.1. Overview

The Large Scale Event scenario described in detail in Deliverable 4.1 [1], foresees using network-external data to predict irregular large-scale events, which may cause the unexpected spikes of activity in the network. In practice, we intend to focus on a specific period in time and a specific location, analyse the data coming from the social media, in particular from Twitter, and learn how they correlate with the network usage data for this period and location. Capturing this correlation will aid to predict unexpected network load and can be employed when planning how to update the status of the 5G network, in order to guarantee high quality of service.

Currently, high-quality network usage data are not available to us. As of December 2016, we are evaluating the first tranche of the WeFi network traffic consumption data collected in the New York area in August 2016 and determining whether they are fit for our scenario.

Therefore, so far, we have focused our research and engineering efforts on processing the data coming from the social media. We have been collecting Twitter data since May 2016. By December 2016, we have collected 65 GB tweets from the New York area in json-format, and around 24 GB of tweets from the territory of Italy. Our ultimate goal is to find large clusters of tweets referring to the events likely to impact the network traffic consumption in the incoming Twitter data. However, first we need to determine what kinds of events are likely to trigger the spikes in the network consumption. Availability of high-quality network usage data is essential for this task. Provided that the network usage data are only under evaluation at the moment, we leave the "event tweet" detection for the future work.

In the current release, we focus on tweet clustering. We provide (i) a model, which given two tweets referring to an event predicts whether they refer to the same event (we will further refer to it as “same-event” tweet pairwise classifier); (ii) a Java pipeline for training/deployment of such model. The following subsections describe the contributions in more detail.

2.1.1.1 Contribution 1. “Same-event” pairwise tweet classification model

Our approach to “same-event” pairwise tweet classification employs the technologies for classifying short text pairs based on usage of relational structural data representations, which we refined earlier in the course of the project (see [4], [5]), and usage of the structured input machine learning algorithms described in D3.1 [6]. We have trained the model on the First Story Detection
(FSD) corpus by [7] that contains 3,035 “event” tweets labeled with 27 events\(^2\) judged as important by the WikiNews\(^1\) portal, e.g. death of Amy Winehouse or the news about the first artificial organ transplant. In our intuition, these events are not likely to cause spikes in usage of the network data in a specific area, however, we believe that a model trained on these data can be reapplicable with minor adjustments to the “event” tweets relevant for the LSE scenario. In order to train the “same-event” binary classification model, we randomly selected 12 events from the FSD corpus, took the subset of tweets annotated with these events and generated 50K random tweet pairs (TRAIN dataset). If both tweets in a pair refer to the same event, we label this instance as positive; we label it as negative, otherwise. For testing, we used the remaining 13 FSD events in the similar manner, generating all possible unordered tweet pairs, 1 million pairs in total (TEST dataset).

We trained the model with an SVM employing a composite kernel that is a sum of Partial Tree Kernel (PTK) and a polynomial kernel. Refer to D3.1 and [5] for more details regarding PTK and combining it with polynomial kernel for classifying the short text pairs. The model classifies the input “event” tweet pairs, \((tweet_1, tweet_2)\), represented as the following triple:

- relational structural representation of \(tweet_1\)
- relational structural representation of \(tweet_2\)
- similarity feature vector

Refer to [4], [5] for the overall high-level theoretical description of the relational structures. Refer to Sections 2.1.3, 2.1.4 for the explanations on how to generate input structures to be used by this particular model.

We trained the classifier on TRAIN and tested it on TEST. It achieved the precision, recall and F1-measure of 97.2, 75.3 and 84.8, respectively, outperforming the similarity feature-based polynomial kernel SVM with precision, recall and F1-measure of 95.38, 69.95 and 80.71, respectively.

2.1.1.2 Contribution 2. RelationalTextRanking Java pipeline

We contribute the RelationalTextRanking Java pipeline that is a flexible pipeline for converting pairs of raw texts (candidate “same-event” tweets in case of LSE scenario) into structured representations enriched with the relational information and serializing them as input data for SVMLight-TK\(^4\) and its Spark-TK extension presented in Deliverable 3.2 [3]. The pipeline is based

\(^2\) According with the Twitters restrictions on publishing Twitter corpora, [7] made available only the ids of the tweets. We used our own script to retrieve them, however, some of them are not available anymore. We have recovered the dataset of around 1,974 unique tweets annotated with 25 topics.

\(^3\) https://en.wikinews.org

\(^4\) http://disi.unitn.it/moschitti/Tree-Kernel.htm
on the Apache UIMA technology\(^5\), which allows creating highly modular applications that analyse large volumes of unstructured information.

**RelationalTextRanking** is a modular pipeline, which is very easy to adjust. We based the pipeline on the NLP processors and infrastructure available within the IKernels group of University of Trento; then we further developed the pipeline when working on the approaches for ranking and classifying text pairs in the first year of the CogNet project \(^4\). No extensive coding effort is needed for adjusting it for a new text type, plugging in new linguistic annotators, devising new types of structural representations, and the changing of the format of the output data.

Figure 2 illustrates the overall structure of the pipeline. **RelationalTextRanking** expects as input a tweet, text\(_i\), and a list of \(n\) other tweets (text\(_{i1}, \ldots, \text{text}_{in}\)). For each tweet from the latter list, we check whether it refers to the same event as text\(_i\). We will further refer to the tweets from this list as candidate "same-event" tweets. At runtime, **RelationalTextRanking** forms pairs \(P=(\text{text}_i, \text{text}_{ij})\) and generates their relational structural and feature vector representations.

The most important **RelationalTextRanking** modules are as follows:

- **System.** This is the entrance point to the pipeline. The module initializes and manages construction of the relational structural and similarity feature vector representations for the text\(_i\), text\(_{ij}\) pair, and their serialization as input data for SVMLight-TK/Spark-TK. We provide several System classes in our pipeline, all of them descendants from the base it.unitn.system.QABase abstract class that incorporates some basic functionality.

![Figure 2 Overall Schema of the RelationalTextRanking pipeline]

\(^5\) [https://uima.apache.org/](https://uima.apache.org/)
• **UIMA pipeline.** The module runs a pipeline of UIMA Analysis Engines (AEs), which wrap linguistic annotators, e.g. Sentence Splitters, Tokenizers, Syntactic parsers, thus converting input text pairs into UIMA Common Analysis Structures (CAS). CASes contain the original texts and all the linguistic annotations produced by the AEs. AEs produce linguistic annotations defined by a UIMA Type System. Additionally, there is an option to persist the produced CASes, and not to rerun the annotators again when re-processing a specific document. Currently, we provide the UIMA AEs based on the (i) components of the state-of-the-art linguistic processing Stanford pipeline [8], (ii) Illinois Chunker [9] that performs shallow text parsing; and (iii) and the ArkTweet linguistic annotators [10] designed specifically for noisy Twitter text processing. The UIMA pipeline takes a pair of texts, \((text_i, text_j)\) as input, and outputs their respective CASes, \((CAS_i, CAS_j)\).

• **Experiment.** The experiment module defines how the input data are converted to their structural relational and feature vector representations. In particular, **Experiment** defines:
  - Which UIMA Analysis engine to use. When the overall pipeline is initialized, the **Experiment** model communicates to the **System** module which AEs to run within the **UIMA pipeline** (Step 1 in Figure 2) and **System** initializes an UIMA Pipeline with these AEs (Step 2 in Figure 2)
  - Which tree structures (e.g. constituency, shallow or dependency) and which **Projector** module to use.
  - Which features to extract in the **VectorFeatureExtractor** module.

The full name of the Experiment to use is defined when launching the System from the command line. We supply a number of Experiments within the pipeline. Users may define their own experiments by implementing the `it.unitn.nlpir.experiment.Experiment` interface.

The Experiment module takes as input a pair of CASes \((CAS_i, CAS_j)\) with all the linguistic representations produced by the UIMA pipeline, and outputs their relational structural representations, \((RelStruct_i, RelStruct_j)\), and (optionally) their respective feature vector, \(FV_{ij}\). The latter incorporates the various characteristics of the \((text_i, text_j)\) pair, e.g. values of the text similarity metrics when applied to \(text_i, text_j\).

• **Projector.** This block is initialized and used by the Experiment module. It enriches the tree representations of \((text_i, text_j)\) with relational links using the information available in \((CAS_i, CAS_j)\), and produces the relational structures \((RelStruct_i, RelStruct_j)\).

• **VectorFeatureExtractor.** This block is initialized and used by the Experiment module. It encapsulates a pipeline of the basic machine learning feature extractors, and produces the text similarity feature vector, \(FV_{ij}\) for the \((text_i, text_j)\) text pair. We have devised a number of basic feature extractors, and additionally, we base some of feature extractors on the components of the state-of-the-art framework for measuring text similarity, DKPro.
similarity\textsuperscript{6} \cite{11}. The user may consult the list of available feature extractors in the it.unitn.nlpir.features.providers\textsuperscript{4} packages.

- OutputWriter. The System module programatically defines which OutputWriter to use. It converts the relational structural representations, \((RelStruct_i, RelStruct_j)\), and the feature vector \(FV_j\) produced by Experiment, into the machine learning instances and serializes the \((RelStruct_i, RelStruct_j, FV_j)\) triple into the training/test files to be consumed by a specific machine learning algorithm implementation. Currently, all the data are serialized in concordance with the SVMLightTK/Spark-TK input format.

In Figure 2, the arrows show the connections and the flow of the data between the components described above. Here, the grey dashed lines correspond to the initialization and finalization steps. The solid yellow lines correspond to the K-th text pair, \((text_i, text_j)\) processing step.

In the “same-event” tweet detection task, the System module is represented by the it.unitn.nlpir.system.classification.TextPairClassification class, which generates the structural representations of two tweets and their textual similarity feature vector and serializes them in the SVMLightTK/SparkTK format.

The Experiment module for the “same-event” tweet detection task is represented by it.unitn.nlpir.experiment.twitterdup.CHPunktVDKProFeatNoESAArkTweetExperiment. It sets up the UIMA pipeline, which embeds Stanford CoreNLP annotators, Illinois Chunker, and ArkTweet tokenizers. The module produces the shallow structures with lemmas of the words at leaf level, with part-of-speech (POS) nodes as their parents. The POS nodes are grouped under the chunk nodes; chunk nodes are grouped under the sentence nodes. Figure 5 demonstrates an example of such structure. The module also initializes a VectorFeatureExtractor that extracts a set of features based on the textual-overlap similarities and tree-kernel similarities between tweet\textsubscript{1} and tweet\textsubscript{2}. We refer the reader to the documentation of the CHPunktVDKProFeatNoESAArkTweetExperiment class for more details.

2.1.2. Download, build and Installation

The tool requires the following prerequisites:

- Java 1.8+
- Apache Maven > 3.3.9. Refer to http://maven.apache.org/install.html for the installation instructions
- Additional DKPro resources for computing semantic Wikipedia and WordNet-based DKPro similarity features. Refer to the https://dkpro.github.io/dkpro-similarity/settinguptheresources/ for the instructions on how to setup the following DKPro resources (Please, remember to set up the DKPRO_HOME environment variable as described in the installation instructions web-page):

\textsuperscript{6} https://dkpro.github.io/dkpro-similarity/
D4.2 – Raw Data Preprocessing, Prediction in NFV, Self-Managed NFV Ecosystem, Network Traffic Classification and Prediction.

- WordNet Lexical Semantic Resource index. Note that if you use only the classes mentioned in this deliverable, you need to download and setup only the WordNet Lexical Semantic Resource index (see Lexical Semantic Resources for Word Aggregation Measures subsection of the DKPro installation instructions). Follow all the official installation instructions, but substitute the original wordnet_properties.xml file supplied within WordNet resource graph archive, with the following file instead: https://raw.githubusercontent.com/dkpro/dkpro-lsr/master/de.tudarmstadt.ukp.dkpro.lexsemresource.wordnet-asl/src/main/resources/resource/WordNet_3/wordnet_properties.xml.

  We do not employ Wiktionary in the pipeline. Therefore, you need to remove the following lines from the resources.xml file (or, alternatively, you may download and install the Wiktionary resources as described in the DKPro installation instructions):

  ```xml
  <bean id="wiktionary-en" lazy-init="true"
    class="de.tudarmstadt.ukp.dkpro.lexsemresource.wiktionary .WiktionaryResource">  
    <constructor-arg value="ENGLISH"/>
    <constructor-arg
      value="${DKPRO_HOME}/LexSemResources/Wiktionary/jwktl_0.15.2_en20100403"/>
  </bean>
  ```

- Wikipedia Explicit Semantic Analysis index. If you want to be able to access to the full range of features available in this pipeline (note: this not needed for training/testing the “same-event” tweet detection model provided within this deliverable), please, download the precompiled the Wikipedia Explicit Semantic Analysis index (see the Explicit Semantic Analysis: Vector Indexes section of the DKPro installation instructions).

  - Apache UIMA. Follow the instructions at https://uima.apache.org/downloads.cgi. Note that you need to install Apache UIMA only if you are planning to modify or recompile the UIMA type system supplied with the RelationalTextRanking. Otherwise, all the relevant UIMA libraries are already included into the dependencies, and you do not need to install anything.
  
  - Git

  Additionally, in order to run the training/prediction on the files produced by RelationalTextRanking, you need to install either SVM-Light-TK\(^7\) or its extension Spark-TK (refer to Section 3.1 of D3.2 [3] for the tool description and installation instructions) that allows parallelizing the classification phase by using the Spark Apache environment.

  After making sure that you have all the necessary prerequisites, install the module as follows.

---

\(^7\) Refer to http://disi.unitn.it/moschitti/Tree-Kernel.htm for the installation instructions
1. **Clone the project repository and set JAVA_HOME variable**

```bash
git clone https://github.com/CogNet-5GPPP/WP4-CSE.git
cd ./RelationalTextRanking
```

Set JAVA_HOME variable.

2. **Build the Maven project**

This is a Maven project, however, one of the required libraries, PTK.jar, is not available in any of the online Maven repositories. We supply it with our distribution and you should import it manually into your maven repository:

```bash
mvn install:install-file -Dfile=lib/PTK.jar -DgroupId=it.unitn.kernels.ptk -DartifactId=ptk -Dversion=1.0 -Dpackaging=jar
```

Then download the dependencies and compile

```bash
mvn clean install
mvn clean dependency:copy-dependencies package
```

3. **Generate the UIMA Annotation Type classes (optional)**

We provide the pre-generated UIMA type classes. They are identified by the UIMA Type System descriptor in desc/PipelineTypeSystem.xml.

In case if you modify the type system for your purposes, you need to regenerate the classes, following the instructions in [https://uima.apache.org/d/uimaj-2.4.0/tutorials_and_users_guides.html#ugr.tug.aae.generating_jcas_sources](https://uima.apache.org/d/uimaj-2.4.0/tutorials_and_users_guides.html#ugr.tug.aae.generating_jcas_sources).

4. **Download the ArkTweet POS-tagger model**

Go to the root folder of your RelationalTextRanking distribution. Execute the following command:

```bash
wget http://www.cs.cmu.edu/~ark/TweetNLP/model.20120919 -P tools/ark-tweet-nlp-0.3.2/
```

2.1.3. **Training**

As explained in Figure 2 and Section 2.1, there can be multiple configurations for generating the training SVMLight-TK files with RelationalTextRanking. In order to generate the training data for the "same-event" tweet detection model, run the following command:

```bash
java -cp target/classes/:target/dependency/*:resources -Xmx5G it.unitn.nlpir.system.classification.TextPairClassification -expClassName it.unitn.nlpir.experiment.twitterdup.CHpunktVDKProFeatNoEAArkTweetExperiment -mode train <other-parameters>
```

The essential <other-parameters> to specify are as follows:

- `questionsPath [String]`: path to the file containing list of unique tweets, `tweet_i`, corresponding to `text_i` in Figure 2. The file should be space-delimited

1. `tweet_i`, unique id
2. \textit{tweet}, raw text

- \textit{answersPath} [String]: path to the file containing a list of candidate "same-event" tweets for the tweets from the list stored in location defined by the \textit{--questionPath} parameter. The file should be space-delimited and contain the following columns:

1. \textit{tweet}, unique id, where \textit{tweet} is the tweet from the file defined by the \textit{--questionsPath} parameter.
2. \textit{tweet} id: current candidate "same-event" tweet id. The system will assemble pairs (\textit{tweet}, \textit{tweet} id) at runtime (they are depicted as \textit{text}, \textit{text} id in Figure 2).
3. 0.0 (only when using the CHPunktVDKProFeatNoESAArkTweetExperiment class. If you are using another Experiment implementation consult its particular Javadoc.)
4. 0.0 (only when using the CHPunktVDKProFeatNoESAArkTweetExperiment class. If you are using another Experiment implementation, refer to its particular Javadoc.)
5. "true" if (\textit{tweet}, \textit{tweet} id) refer to the same event, "false" otherwise
6. raw text of \textit{tweet} i

- \textit{candidatesToKeep} [Integer]: Number of the candidate "same-event" \textit{tweet} id from the file defined by \textit{--answerPath} to consider per \textit{tweet}, when generating the (\textit{tweet}, \textit{tweet} id) pairs.
- \textit{outputDir} [String]: Output directory to write the generated files to. The system will generate the svm.train file with the SVMLight training data, and the svm.train.relevancy file with the ids of (\textit{tweet}, \textit{tweet} id) aligned with svm.train line-by-line
- \textit{filePersistence} [String]: Path to the folder into which to persist the UIMA CASes generated by the system. If this parameter is not set, the CASes will not be persisted.

2.1.3.1 Training toy example

Twitter does not allow publishing Tweets texts as a corpus\textsuperscript{8}, therefore we cannot provide a sample toy dataset with tweet texts to run the pipeline upon. Instead, we include the data files with the simplified training data from the SemEval Community Question Answering (CQA) competition, Task A\textsuperscript{9}. Here, we will still generate the training files for learning a classifier to predict a label for a pair of texts, however, in this case the texts in the pair will be a forum question (from data/semeval2016/train/questions.txt) and its candidate answer (data/semeval2016/train/candidates\_75w.txt). The pair is labeled as "true" if the candidate answer correctly answers the question and "false" otherwise.

Go to the root folder of the pipeline distribution and run the following command:

```
java -Xmx5G -cp target/classes/:target/dependency/*:resources it.unitn.nlpir.system.classification.TextPairClassification -questionsPath data/semeval2016/train/questions.txt -answersPath
```

\textsuperscript{8} [https://dev.twitter.com/overview/terms/policy.html#f-be-a-good-partner-to-twitter](https://dev.twitter.com/overview/terms/policy.html#f-be-a-good-partner-to-twitter), Section F.2

\textsuperscript{9} [http://alt.qcri.org/semeval2016/task3/](http://alt.qcri.org/semeval2016/task3/)
The code above will generate the following files in the `data/examples/toy-semeval-example` folder: (i) `svm.train`: SVMLight-TK input file; (ii) `svm.train.relevancy`: file with the ids of the question/answer pairs included into `svm.train`.

You can use SVM-Light-TK/Spark-TK to train a model. Instructions on how to obtain, install and set the Spark-TK tool are provided in D3.2. Once you have installed it, go to the SVMLight/Spark-TK folder and run the following command:

```
./svm_learn -t 5 -F 3 -C + ${PIPELINE_PATH}/data/examples/toy-semeval-example/svm.train ${PIPELINE_PATH}/data/examples/toy-semeval-example/svm.model
```

Set the `${PIPELINE_PATH}` environment variable to the RelationalTextRanking location on your machine. Refer to SVMLight/Spark-TK documentation for the meaning of the input parameters of `svm_learn`.

### 2.1.4. Deployment

Running the pipeline for converting the new data into the SVMLight-TK test files does not differ from running it for generating the SVMLight-TK training files.

The only differences are that you need to set the `--mode` parameter value to `--test`, and you can put any dummy label (true or false) into column 5 in the `--answersPath` file. The pipeline will produce the `svm.test` and `svm.relevancy` files with the SVMLightTk representations of the new data as machine learning instances and their ids, respectively.

#### 2.1.4.1 Deployment example

We provide a deployment example with simplified development data of the SemEval-2016 CQA competition, Task A.

```
java -Xmx5G -cp target/classes/:target/dependency/*:resources it.unitn.nlpir.system.classification.TextPairClassification -questionsPath data/semeval2016/dev/questions.txt -answersPath data/semeval2016/dev/candidates_150w.txt -outputDir data/examples/toy-semeval-example -filePersistence CASes/toy_semeval_train -candidatesToKeep 10 -mode train -expClassName it.unitn.nlpir.experiment.twitterdup.CHPunktVDKProFeatNoESAExperiment
```

Then, if you have Spark-TK installed you may use the following command to run prediction using the model that you have trained following the instructions in the “Training example” subsection.

```
./Spark-TK.sh ${PIPELINE_PATH}/data/examples/toy-semeval-example/svm.test ${PIPELINE_PATH}/data/examples/toy-semeval-example/svm.model ${PIPELINE_PATH}/data/examples/toy-semeval-example/svm.output 10 <scala-version>
```

Refer to section 3.1.4 of D3.2 for the instructions on how to set the `<scala-version>` parameter.
You will find the resulting predictions in the `${PIPELINE_PATH}/data/examples/toy-semeval-example/svm.output` file.

2.1.4.2 "Same-event" pairwise tweet classification module demo

In this deliverable, we provide the "same-event" pairwise tweet classification model trained on the FSD corpus located in:

`${PIPELINE_PATH}/data/twitter/fsd_model/svm+.model`

We also provide an interactive demo, which reads raw "event" tweet text pairs from the command line input, and predicts whether they refer to the same event.

Before running the demo, you need to build the SVMLight-TK library. Set the `JAVA_HOME` environment variable to the path of your actual Java distribution. Then, go to the root folder of the pipeline distribution and execute the following commands:

```
cd tools/SVM-Light-TK-1.5.Lib
make
```

Return to the folder where you keep the RelationalTextRanking distribution.

The command for running the demo is:

```
java –cp target/classes/:target/dependency/*:resources -Xmx5G -Xss512m it.unitn.nlpir.system.classification.TextPairClassificationDemo -svmModel <path-to-the-pretrained model> -expClassName <experiment-module-name-used-when-training-the-model>
```

In order to run the demo with the pretrained model that we provide in the current release, run the following command:

```
java –cp target/classes/:target/dependency/*:resources -Xmx5G -Xss512m it.unitn.nlpir.system.classification.TextPairClassificationDemo -svmModel data/twitter/fsd_model/svm+.model -expClassName it.unitn.nlpir.experiment.twitterdup.CHPunktVDKProFeatNoESAArkTweetExperiment
```

You will see the prompt asking you to enter two tab-delimited tweets. Enter them and hit "Enter". The system will run prediction on them using `data/twitter/fsd_model/svm+.model`. The output of the system is the distance to the SVM separation hyperplane. If it is above zero, the tweets refer to the same event; otherwise, they do not. The absolute value of the distance is proportional to the degree of confidence of the classifier.

Note that when you enter the first pair of Tweets, the pipeline might be slow because it must initialize all the annotation engines. It will be faster for all the subsequent inputs.

Figure 3 and Figure 4 demonstrate the sample output of the system for the following three tweets:

- **Tweet1.** _Spongebob makes his way down 6th Ave during the #Macy\'s ThanksgivingDayParade_

- **Tweet2.** _Look who it is! Our friend SpongeBob all the way from Bikini Bottom #Macy\'sParade_
• Tweet3. nyc has put up with 90 years of a brand sponsoring a three hour commercial for other brands in the middle of 6th avenue

The model correctly recognizes that Tweet1 and Tweet2 refer to the same event and outputs the prediction 0.11415 for them (see Figure 3); it recognizes that Tweet1 and Tweet3 refer to different events and outputs -0.10551 in Figure 4. The demo outputs the generated relational structural tree representations into console for information purposes. The structures may be visualized with a number of existing online tools (e.g. ironcreek.net/phpsyntaxtree/?) Figure 5 demonstrates a visualization of the relational structure built for Tweet2 when converting (Tweet1, Tweet2) into a machine learning instance.

Figure 3 System output for Tweet1 and Tweet2 (same event)

Figure 4 System output for Tweet1 and Tweet3 (different events)

Figure 5 Visualization of the Tweet2 relational structure built by the pipeline
2.1.5. Development status

In this deliverable, we have contributed the code of the pipeline for the advanced text pair classification and contributed the prototype model for the “same-event” tweet pairwise classification.

The future releases foresee the following:

- Adding Spark support to the RelationalTextRanking
- Further improving the “same-event” tweet pairwise classification model
- Devising a more advanced tweet pairwise clustering technique, which will employ the pairwise “same-event” tweet classification technique as a component.
- Once the telecom data are available, we will devise algorithms for correlating the peaks of the telecom network data usage with the data from the Twitter streams.
2.2. TSDP: Time Series Data Pre-processing

2.2.1. Overview

One of the keys to the successful application of machine learning methods is the transformation of the input data into an adequate representation, that is, into a format that carries as much information as possible about the task to be solved. One of the most important characteristics of data collected within the network infrastructure, and in particular the data collected for the Optimized Services in Dynamic Environments use case, is the fact that it is in the time series format. This means that valuable information can be extracted by examining the data from the time domain perspective.

TSDP (Time-Series Data Pre-processing) is a pre-processing tool for time series data. Specifically, TSDP processes a time-series data set to transform it into a machine-learning-readable form, in a way that the time-series structure can be best leveraged. The input data to be processed by TSDP should be in the following format:

[YYYY-MM-DD HH:MM:SS] <feature_value ... feature_value>

The output file is the result of removing the timestamp and further processing according to the input arguments. Specifically, TSDP can perform a series of useful tasks:

- **Concatenation**: input data instances can be concatenated together in strings of consecutive events. This allows any machine learning practitioner to effortlessly benefit from the information that can be extracted from the temporal domain of the data.

- **Aggregation**: input data instances can be aggregated into the mean of events happening in certain time intervals. This is useful if the granularity of the raw data is higher than necessary. In these cases, aggregation can yield significant improvements in efficiency while sacrificing little performance.

- **Standardization**: the data can be standardized to zero mean and unit variance if desired.

- **Label binarization**: TSDP can turn continuous target values into binary labels, using a threshold specified by the user.

2.2.2. Download, build and installation

1. **Dependencies**:

   - Python 2.7
   - Numpy: [http://www.numpy.org](http://www.numpy.org)
     - Standard python library for scientific operations.
     - data analysis tools for python
2. Clone TSDP

Use the following command to clone TSDP from the repository.

```
git clone https://github.com/CogNet-5GPPP/WP4-CSE.git
```

Then, go into the project directory.

```
$ cd ./WP4-CSE/TSDP
```

2.2.3. Deployment

TSDP is a Python script and does not need to be compiled or installed. The command to run TSDP is as follows:

```
$ python tsdp.py -i|--input=<input file> -o|--output=<output file>
[-c|--concatenate=<integer> -a|--aggregate=<integer> -s|--std
-t|--thr=<float>]
```

- input: Input file path
- output: Output file path
- concatenate: If present, each row of the output data set consists of n concatenated rows of the original data set.
- aggregate: If present, the rows of the output data set are the result of aggregating N rows from the original data set and taking their mean.
- thr: If present, the labels are binarized according to the given threshold (values above are transformed into ones, the rest into zeroes).

2.2.4. Preprocessing of the time series data

Before using time series as inputs to the CSE module, we performed a series of techniques to clean and organize the data. Then, we used other techniques to select only the time series that can be exploitable by the CSE.

Firstly, the data received through REST API from our monitoring service was in a raw JSON form (see Figure 1). The first operation is to filter the data and reorganize it as a data frame table as in Figure 1. The output of this phase is data structured as time series with multiple variables (columns). The variables represent all the metrics that are collect by the monitoring agent. They are the information that we have on the system. The manipulation of these variables can extend/augment the information we perceived on the system being monitored. The process to do so is called: feature engineering.

From these variables we generate another set of variables (or features) using two techniques:
1. The combination of two or more variables into one, e.g. \((\text{CPU/RAM})^2\)

2. Applying a function to only one variable to stress its behaviour e.g. stressing the evolution of a variable \(\text{CPU}^3\)

One important aspect of the data preparation phase is the ability to filter out the data that cannot be used by the machine learning algorithms. For example, white noise, or other data with no correlation. The technique used is the autocorrelation; it allows us to study if the time series data are (1) forecastable and (2) to identify for how many time steps we can see in the future. In the figure 1, the autocorrelation identified the correlation between the previous step and the next step, the results are that there is a correlation between \(x(t)\) and \(x(t-1)\) up to 80%, but between \(x(t)\) and \(x(t-2)\) the correlation drops to 60%.

![Autocorrelation](image)

**Figure 6 The autocorrelation function of the load**

The feature engineering is an empirical and iterative process. The features that yields the best output in terms of cost function are kept. Then the data are normalized and rescaled to \([0-1]\) interval (see equation 1).

\[
X_j = \frac{X_j}{\sum_{i=1}^{N} X_i} \quad (1)
\]

- \(X_i\) is the value of the metric \(X\) at step \(i\)
- \(N\) is the total occurrence of the metric \(X\)
The second step complementary to the feature engineering is the dimensionality reduction. The aim of this phase is to reduce the dimensionality of the data by keeping only the most relevant components that capture the system behaviour. In the literature multiple techniques are used such as high correlation filtering, Low variance filter, backward feature elimination, etc. For this work we settled on PCA – Principal Components Analysis, a technique that uses an orthogonal transformation on the data to create new vectors uncorrelated that capture the most variance in the data set. We applied this technique to 24 variables and the results, as shown in Figure 2, was 7 Principal Components (PC). These PCs are abstractions of the input variables.

<table>
<thead>
<tr>
<th>sdev</th>
<th>varprop</th>
<th>cumprop</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard deviation</strong></td>
<td><strong>Proportion of Variance</strong></td>
<td><strong>Cumulative Proportion</strong></td>
</tr>
<tr>
<td>PC1 1.726645</td>
<td>0.425407</td>
<td>0.425407</td>
</tr>
<tr>
<td>PC2 1.306952</td>
<td>0.244017</td>
<td>0.670425</td>
</tr>
<tr>
<td>PC3 1.049037</td>
<td>0.157211</td>
<td>0.826636</td>
</tr>
<tr>
<td>PC4 0.773208</td>
<td>0.085407</td>
<td>0.912043</td>
</tr>
<tr>
<td>PC5 0.606810</td>
<td>0.052603</td>
<td>0.964646</td>
</tr>
<tr>
<td>PC6 0.439620</td>
<td>0.026861</td>
<td>0.991507</td>
</tr>
<tr>
<td>PC7 0.243830</td>
<td>0.008493</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

For our first implementation, we used the threshold on the cumulative variance to 80%, which means that we can limit the features into: PC1, PC2 and PC3. The objective of the machine learning is to learn the patterns within the time series in order to forecast their evolution through time. The Machine Learning approach is based on a special type of Recurrent Artificial Neural Networks termed LSTM for Long Short Term Memory. The LSTM approaches were introduced in 1997 [18], their main advantages are to retain information over many time intervals. They have been used successfully in image recognition, translation, language representation, driverless cars,
The RNNs have distributed hidden states that allow them to store information about the past and update the information in a non-linear way. They leverage the concept of associative memory and can identify a previously seen pattern from a new distorted version. This is particularly useful to model stochastic dependencies in Time series [20]. We built our LSTMs using the Keras library based on Google’s TensorFlow [21].

![Simple LSTM Cell with a recurrent connection](image)

**Figure 9 Simple LSTM Cell with a recurrent connection**

### 2.2.5. Machine learning for time series

In order to forecast metrics behaviour in the MMCC scenario, we based our approach on the following axioms at time $t$:

- Each metric exhibit a non-null correlation, i.e. a pattern exists in the TS
- Correlation between different metrics and other more abstract features exists in the network (e.g. CPU, network utilization)
- Network centric metrics can affect the service quality and/or the whole end-to-end service

#### 2.2.5.1 The learning phase:

The objective is to infer the future behaviour of the service and of the underlying network at time $t+1$ (i.e. in the next step). We have at time $t + 1$ a forecasted Matrix $Y$ defined as:

$$Y = \begin{cases} 
    PC1(t + 1) + \epsilon_1 \\
    PC2(t + 1) + \epsilon_2 \\
    PC3(t + 1) + \epsilon_3 
\end{cases}$$

We adopted a supervised learning approach. The supervised learning requires the labelisation of input data, each entry has a corresponding label. For time series, in order to learn the relation between the past and the future (Graph in red figure 6), we set as inputs the graph in blue (Figure 6) and the target the same graph but shifted by one step (Graph in green Figure 6).

The LSTM then computes the lowest cost function between the past and the future. The resulting model, i.e. nodes coefficients represents how we can derive from $x(t + 1)$ from $x(t)$. 

---

**CogNet**

**Version 1.0**

**Page 26 of 45**
2.2.5.2 The scoring phase

Once the model learnt, the prediction is performed in real-time. These predictions are used as additional information for managing the network, i.e. preventing future degradations. However, the limit of this approach is that the model can be outdated and thus should be constantly updated.

2.2.5.3 Pitfalls

Traditional machine learning methodologies rely on dividing the data set into 2 or 3 subgroups. One large group (generally 70% or more) as the dataset for the training, and the others as data sets for test and evaluation. This methodology allows us to test the accuracy and precision of our model. However, in data structured as time series, dividing arbitrarily the data set may break the structural dependencies of the observation. For example, breaking a cosine function from $0$ to $\frac{2\pi}{3}$ as training set and the last $\frac{\pi}{3}$ as the test set cannot converge the cost function and thus cannot learn and generalize. To avoid this problem one should be careful to take into account a relatively wide window of observation depending on the use case and the data set.

Another pitfall that may slow down the learning process is that for time series data, the inputs should be retransformed as percentages. This means that the value of $X$ at time $t$ is the percentage of its evolution from $t-1$ and not its absolute value. This preprocessing operation allows the ANN to learn the time series evolution not the series themselves.

2.2.5.4 Storage

In our work, the time series are stored in three different formats:

1. CSV – Comma Separator Value: Simplest way to store data, it is effective for small data and data preparation phase. It can be easily analysed by a human operator.
2. SQL: It is used as the back end of our system, can store high dimensionality data. However this solution might be a hurdle when scaling out to millions of read/write operations per second.

3. InfluxDB: a high availability throughput database specialized for time series storage, it is used by the monitoring system Monasca.

### 2.2.6. Example

The repository contains an example data set in the following path:

```bash
eample/toy_data.txt
```

Below is an example of the output obtained after running TSDP on these data.

First, we print the first ten lines of the file to reveal its structure (Figure 11).

![Figure 11 Time series data to be processed by TSDP.](image)

The first column is the timestamp, the next three are the input feature and the last one is the target value.

We run TSDP so as to remove the timestamp, standardize the variables and binarize the target value. We also choose to concatenate three events and aggregate them over periods of three events. The corresponding command can be seen in Figure 12.

![Figure 12 Command to run TSDP.](image)

The result can be observed in Figure 13. Each row now has ten elements (nine variables and a binary target value), and represents the mean value of three consecutive rows.

![Figure 13 Output of TSDP](image)
2.2.7. Development status

TSDP currently offers functionalities exploited within the Follow the sun scenario. So far, the experiments carried out in the context of this scenario have been of a preliminary nature. In the coming months of the project we intend to significantly increase the complexity of the analysed data, in order to enable the exploitation of the machine learning methods developed for the Follow the Sun scenario in an industrial environment. Therefore, we plan enhance this component to cater to the needs that arise in this more complex scenario, as well as to add Spark support.

TSDP is also used in the context of MMCC scenario. The bulk of the work have been done in the pre-processing part.
2.3. Data pre-processing for demand prediction

2.3.1. Overview

As described in D4.1, the Urban Mobility Awareness (UMA) scenario focuses on Recurrent situational context in which we aim to predict the regular but different recurrent patterns in cities that will impact the network demand and hence, a smarter network resource allocation can be developed based on these recurrent patterns correlated with their expected network demand. Traditional network resource management systems cannot tackle the impact of the variation of recurrent activities in cities on provisioning network resources since external real scenarios cannot be captured by network monitoring systems. A more advanced cognitive system needs to be deployed to address the challenge by tracking multiple levels of scenarios in cities while predicting variability in resource demands based on the various recurrent activities in cities. This scenario focuses on predicting network demand according to parameters such as location, time and specific service demand from specific users or user group. In this deliverable, we report our algorithms developed for detecting temporal functional regions that varies over space and time that is representing the pre-processing procedures before the correlation with the real network demand datasets that is expected to be available for the consortium soon.

2.3.2. Download, build and Installation

1. Dependencies:
   - Python 2.7
   - Numpy: http://www.numpy.org
     - Standard python library for scientific operations.
   - Git

2. Clone the repository:

The libraries associated with test data can be cloned by:

```
git clone https://github.com/CogNet-5GPPP/WP4-CSE.git
cd ./WP4-CSE/DP
```

This work assumes that users have already install all the python and libraries requested in the work, which include “sklearn”, “pandas”, and “geojson”. Based on that, the pre-processing consists of five steps:

1. Data filtering and splitting based on a selected time interval – In this step, we pre-process raw data and map it into a suitable format that can be consumed by the selected clustering algorithms in the next step. The outcome of data filtering is a set of Spatial-temporal Functional Matrices, each of which covers a given time-slot. Every Spatial-temporal Functional Matrix illustrates the probability of each physical area belongs to
functional categories. The pre-processing work can be decomposed into Activity Categorization, and Time-slot Aggregation as follows:

2. **Activity Categorization**: The visited location of each check-in is classified into 9 categories: Entertainment, Education, Night Life, Recreation, Social Services, Residence, Shopping, Travelling, and Eating, as used in Foursquare’s applications. These 9 activity categories will act as the functionalities of regions in our work. This pre-processing step will not only decrease the sparsity problem significantly when feeding the data to clustering techniques but also enable more meaningful functionalities of regions. In particular, this categorization maps 532 different keywords in the dataset into 9 labels as functional regions.

3. **Time-slot Aggregation**: The daily records are then split into 24, 12, 6, 3 time-slots, each of which lasts for 1, 2, 4, and 8 hours respectively. In general, the smaller the slot, the finer the region’s functionality can be captured but with more sparsity challenges. The step aims to study the tradeoff between sparsity of the dataset and preservation of the meaningful functionality of a region.

4. **Clustering and performance evaluation** – It performs clustering based on a selected technique, and evaluates the performance of the solution based on varied number of clusters. The silhouette coefficient is applied as the evaluation metric that is calculated based on the Euclidean pairwise difference of between and within cluster distances. The silhouette coefficient is not dependent on any existing labels, whose value is bounded between -1 for incorrect and +1 for highly dense clustering while value around 0 indicates overlapped clusters. The setting offering the largest silhouette coefficient value will be selected and generate the final clustering result.

5. **Functionality identification** – The output of the clustering is consumed this steps. It calculates the weight of each keywords on every cluster – one functional region. This is achieved by summarising the number of check-ins on a given keyword in one functional region and then divided by the total number of check-ins in whole data set on that keyword. The keyword with the largest weight is picked up as the functionality of a cluster.

**2.3.3. Deployment**

All the data that are consumed in pre-processing are given in the folder “data” while the generated results are given in "output". To complete the five steps above, invoke the code for filtering and splitting and then perform clustering:

```python
python 1_dataPreProcessing.py
python 2_clustering_functional_regions.py
```

Please note the time interval for splitting can be set in the code for filtering and splitting, and clustering work can be extended for adapt to more clustering techniques. One part of log information for clustering and the result for hierarchical clustering are illustrated in Figure 1 and 2 respectively.
After that, the R code for functionality identification needs to be invoked by

```
Rscript 3_area_clustering_withkeygen.R
```

Please note to get the best results, the “nFeature” valuable in the code may need to be adjusted to adapt to number of clusters identified in the previous steps if the value is larger than 13.

```
dealing with file: output_04AM_08AM.csv
k= 2  performance= 0.427779158486
k= 3  performance= 0.498369343126
k= 4  performance= 0.553230494182
k= 5  performance= 0.556583621216
k= 6  performance= 0.579089998257
k= 7  performance= 0.595878457393
k= 8  performance= 0.620310226969
k= 9  performance= 0.633291653834
k= 10  performance= 0.638734214193
k= 11  performance= 0.640746958793
k= 12  performance= 0.63473941324
k= 13  performance= 0.62923288453
Hierarchical clustering max performance value= 0.640746958793  max value index= 11
```

```
dealing with file: output_04PM_08PM.csv
k= 2  performance= 0.485663181349
k= 3  performance= 0.392964296493
k= 4  performance= 0.44583965349
k= 5  performance= 0.470651449933
k= 6  performance= 0.495803957297
k= 7  performance= 0.49990874737
k= 8  performance= 0.492016535762
k= 9  performance= 0.493376023037
k= 10  performance= 0.507304876382
k= 11  performance= 0.515045596898
k= 12  performance= 0.52154071658
k= 13  performance= 0.469818715629
Hierarchical clustering max performance value= 0.52154071658  max value index= 12
```

Figure 14 Functional region clustering.
2.3.4. Development status

The above libraries have been applied in [12] to validate our solution on data preparation. It may be enhanced as the further investigation in the area by the IBM team based on new requests and data available in the project. Development work will adapt to the latest achievement on the theoretical research.

Figure 15 Clustering results with varied number of clusters when applying Hierarchical clustering.

Figure 15 results are generated for exploring optimum number of clusters k for the 4 Hours interval resulting in 6 time slots. As it can be observed, the optimum number of clusters for Hierarchical clustering for the 4 Hours interval are k=13, represented by a dashed vertical line.
2.4. MoSeS (Model Selection Service)

2.4.1. Overview

One of the most important challenges of the application of machine learning is selecting a proper model to solve the problem at hand. An important theoretical result, known as "No free lunch" [2], establishes that no particular algorithm is better than any other in a general sense. This implies that given a new problem (that is, a new data set) there is generally no good reason to choose any algorithm over another, other than data format and structure, intuition or insight gained via exploratory analysis.

Therefore, when applying machine learning in practice it is important to devise an effective procedure to select the most appropriate model to use. On one hand, it is necessary to choose a learning algorithm based mainly on the complexity of the task and efficiency constraints. On the other, it is necessary to tune the hyperparameters of the chosen algorithm. This is usually done using cross validation, which basically consists in dividing the data set into various chunks to get a reliable estimate of how the model will perform in practice. Different sets of hyperparameters can be tested using different methods, such as grid or random search.

This component trains and evaluates various machine learning models on the input data set using cross validation and grid search. The model with the highest area under the receiver operating characteristic (ROC) curve is saved to disk. Currently, the MoSeS component evaluates random forests and support vector machines (SVM).

This component is designed to be usable within an infrastructure management pipeline. More specifically, this component exposes an external interface and can thus be easily integrated into the Monitoring, Analysis, Planning and Execution (MAPE) Loop that we are considering in the CogNet architecture. The rest of the components can interact with MoSeS as they would with any other command line tool.

2.4.2. Download, build and installation

1. Dependencies:

   • Python 2.7

   • Numpy: [http://www.numpy.org](http://www.numpy.org)
     - Standard python library for scientific operations.

   • Git

   • This software was tested on Ubuntu 14.04, although it should work on any system where Python 2.7 is installed.

2. Clone MoSeS

Use the following command to clone MoSeS from the repository.
git clone https://github.com/CogNet-5GPPP/WP4-CSE.git

Then, go into the project directory.

$ cd ./WP4-CSE/moses

2.4.3. Deployment

MoSeS is a Python script and does not need to be compiled or installed.

The command to run MoSeS is as follows:

```bash
python moses.py -i|--input=<input file> -o|--output=<output file> [-k|--kwargs=<model>:<value>,<model>:<value>]+ -c|--conf=<path> -s|--std -v|--verbose
```

- `file`: Input file path
- `output`: Output file path
- `kwargs`: key-value arguments for the machine learning models. Key-value pairs are separated by commas, while the key and the value of each pair are to be separated by a colon. Currently, the following arguments (equivalent to those in scikit-learn) are supported:
  -- Random forests: `n_estimators`, `max_features`, `max_depth`, `min_samples_split`, `random_state`
  -- Support vector machines: `c`, `kernel`
- `conf`: Configuration file path. Run script with `--help` for more information.

One of the most important steps when using MoSeS is the specification of the hyperparameters to be tested. This can be done directly through the command line via key-value pairs or using a configuration file. For instance, if one wishes the random forest model to be tested with 20 trees and a maximum depth of 10, the following command should be used.

```bash
Python moses.py -i <input-file> -o <output-file> -k n_estimators:20,max_depth:10
```

These parameters can also be specified using a configuration file. Each line of the file should indicate the corresponding machine learning model and a key-value pair in the following format:

```bash
<model> <argument> ( <lower>,<upper>[,<stride>] ) | [value]{,value}
```

- `model` specifies the machine learning model that the argument corresponds to. Currently supported models:
  -- `rf` (random forest)
  -- `svm` (support vector machine)
- `argument` specifies the argument to be set.

The values to be tested can be specified in one of two ways.

- `(<lower>,<upper>[,<stride>])` specify the set of values to explore for the specified argument. The first two specify the (inclusive) lower and (non-inclusive) upper bounds of the interval, and the third (optional), the stride.
- `[value]{,value}` (python list syntax with no whitespaces). The list of values to be tested.

Example:
rf n_estimators 10,50,10

Informs the script to test values 10,20,30,40 of the Random Forest parameter n_estimators. A more complex example is described below.

2.4.4. Example

The repository contains an example data set in the following path:

text

Figure 16 shows an example of the output obtained after running MoSeS on these data.

Figure 16 An execution example of MoSeS

The script is run in verbose mode, standardizing the data and using moses.conf as configuration file. If run on verbose mode, MoSeS shows the parameters of each evaluated model and the scores obtained in the cross validation process.

The contents of the employed configuration file used in the example above are displayed in Figure 17.
2.4.5. Development status

This component currently provides some of the features whose need arose during the development of the Follow the Sun and the Massive Multimedia Content Consumption scenarios in the context of the Optimizing Services in Dynamic Environment use case. Both scenarios tackle network management problems as supervised machine learning tasks (noisy neighbor detection and traffic classification), and MoSeS helps us to find models that perform well in solving these problems. In the future we plan to incorporate more machine learning models and more flexibility and efficiency to choose the hyperparameters, as well as to add Spark support.
2.5. Connected Cars

2.5.1. Overview

In the Connected Cars scenario, the network employs machine learning and smart antennas to dynamically adapt the network coverage based on nodes (vehicles in this case) location. This is considered as a cross-layer network management scenario. The software module described here corresponds to the network management part, which was previously described in Deliverable 4.1.

In this software module, a ray-tracing algorithm was developed for the insertion of the propagation loss model resulting from spreading, molecular absorption, scattering and reflection effects in the terahertz-band communication.

First, consider a square area with dimensions of x and y meters and number of n × m tiles. The ray tracing algorithm proceeds with the following four steps: 1) reflection/scattering points are determined, as well as mirror placement; 2) placement of the transmitter (tx); 3) computation of the incident angles based on the defined rays; 4) computation of the referred metric according to the modelling presented in Section II for each tile of the area.

Dielectric mirrors are modelled based on their refractive properties. The refractive index (nt) of the dielectric mirrors is 3.418. These mirrors consist of four 63μm thick layers of high-resistant silicon. This property can easily be added to the Fresnel reflection coefficient. Therefore, the rays that hit the mirrors are attenuated based on their refractive index.

Two approaches are compared, static and adaptive coverage. In the first, the usual antennas are used with no adaptive beamforming and no mirrors. In the latter case, the antennas incorporate the adaptive beamforming and are able to point to the receiver directly utilising the dielectric mirrors for signal reflections.

![Diagram](image)

**Figure 18** Scenario used for distance analysis. The antenna is placed in the central-top position of the area. Both are going to account for the adaptive or static coverage respectively.
For vehicle-to-infrastructure scenario, the antennas are placed on the central-top position of the area, mimicking a roadside infrastructure, Figure 18. Each metric is going to be calculated for each tile of the scenario. For vehicle-to-vehicle scenario, a transmitter (Tx) is going to be placed in the centre of the area and the receiver (Rx) a few meters away, Figure 19.

**Figure 19 Scenario used for frequency analysis. The Tx is place in the center of the area and the Rx a few meters away.**

### 2.5.2. Download, build and Installation

#### 1. Dependencies

The software has the following dependencies:

- Python 2.7
- Numpy: http://www.numpy.org
  - Standard python library for scientific operations.
- Sympy
  - Python library for symbolic mathematical operations.
- Matplotlib
  - Python library for 2D and 3D plotting.
- “Abs_co_1_H2O.txt”
  - File containing water vapour coefficients used to compute losses in the transmitted signal.
- Git

#### 2. Clone the Connected cars repository

Use the following command to clone the Connected cars repository.

```
git clone https://github.com/CogNet-5GPPP/WP4-CSE.git
```
2.5.3. Deployment

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

```python
python CogNet_D42_TSSG_mtaob_v1.py
```

Inputs: The inputs can be formatted inside the code. They are:
- F: frequency
- Txi: transmitter position
- Rxii: receiver position
- Lxi: dimension x of an area
- Lyi: dimension y of an area
- N_rays: number of rays

Output:
A matrix, or multiple, of power distribution over the defined area for a spatial analysis. A frequency analysis can also be used. Different spatial types of output are:
- power_matrix: Matrix of power in each tile of the area
- sca_matrix: Matrix of scattering losses in each tile of the area
- reflection_matrix: Matrix of reflection losses in each tile of the area
- mirror_matrix: Matrix of power now using mirrors for each tile of the area

2.5.4. Example

A vehicle to vehicle scenario and a vehicle to infrastructure scenario are implemented in a small cell network with dimension of 20 by 10. The receiver neglected in this analysis, in which is shown that data can be obtained in each tile of the area. This is necessary to make the antenna of the vehicle synchronize with the mirrors located in the infrastructure, and therefore, re-position themselves cooperatively towards more received power.

The example can simply be run as

```python
python CogNet_D42_TSSG_mtaob_v1.py
```

The resulting output is as shown in Figure 20 and Figure 21.
Figure 20 The output after running the software

Figure 21 shows the graphical output of the example after running the software. This output is, again, correspondent to the power matrix of the small cell. The configuration of the mirrors, the antenna, and the position of the vehicle can change dramatically the output.
2.5.5. Development status

The CogNet team is currently extending the presented method to advance towards the following goals:

1. A more complex data set
2. Mobility data from the city of San Sebastian
3. Integration with the CogNet smart engine
4. Development of signalling protocols for mirror cooperation
3. Conclusions

In this deliverable we have presented some of the components being developed for their integration in the CogNet Smart Engine, the main building block proposed by CogNet to enable autonomic 5G network management. The components described here are intended to bridge the gap between theory and practice in machine learning, and have been developed to cater to the needs of the different scenarios and use cases being addressed, previously described in deliverable D4.1 [1].

During the rest of the project we intend to enhance these components further in order to support a wider array of functionalities, as well as to add Spark support to make sure they are able to process large amounts of data.
References


