

INTELCOMP PROJECT
A COMPETITIVE INTELLIGENCE CLOUD/HPC PLATFORM FOR AI-BASED STI
POLICY MAKING
(GRANT AGREEMENT NUMBER 101004870)

Topic Modeling Service. D3.5.

Deliverable information	
Deliverable number and name	D3.5. Topic Modeling Service
Due date	Jun 30, 2022
Delivery date	Sep 7, 2022
Work Package	WP3
Lead Partner for deliverable	UC3M
Author	José Antonio Espinosa-Melchor (UC3M) Lorena Calvo-Bartolomé (UC3M) Jerónimo Arenas-García (UC3M)
Reviewers	Haris Papageorgiou (ARC) David Gago (SEDIA)
Approved by	Cecilia Cabello (FECYT)
Dissemination level	Public
Version	1.2

Table 1. Document revision history

Issue Date	Version	Comments
Jun 17, 2022	0.1	Initial version with sectioning structure
Jul 19, 2022	0.9	Version ready for internal review (scalability and coherence analysis pending)
Aug 12, 2022	1	Reviewed version (scalability and coherence analysis pending)
Aug 28, 2022	1.1	Scalability and coherence analysis included. Conclusions and future work included. Ready for final review of those sections.
Sep 7, 2022	1.2	Feedback from reviewers included Final version ready for submission Cross-references in MS Word format

DISCLAIMER

This document contains description of the **IntelComp** project findings, work and products. Certain parts of it might be under partner Intellectual Property Right (IPR) rules so, prior to using its content please contact the consortium coordinator for approval.

In case you believe that this document harms in any way IPR held by you as a person or as a representative of an entity, please do notify us immediately.

The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated in the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

The content of this publication is the sole responsibility of **IntelComp** consortium and can in no way be taken to reflect the views of the European Union.

The European Union is established in accordance with the Treaty on European Union (Maastricht). There are currently 27 Member States of the Union. It is based on the European Communities and the member states cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice and the Court of Auditors.



(<http://europa.eu.int/>)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101004870.

CONTENTS

Disclaimer.....	3
Acronyms	8
Executive Summary.....	9
1. Introduction	11
2. Objectives.....	13
3. Topic modeling.....	14
3.1. Description of the Pipeline supported by the toolbox	16
3.2. Description of the Pipeline supported by the toolbox	17
3.3. Topic Modeling technologies incorporated in the TMT	19
3.3.1. Mallet	20
3.3.2. Spark MLLIB.....	21
3.3.3. Neural Topic Models	22
3.3.4. Construction of Hierarchical Topic Models.....	26
3.4. Topic Modeling technologies incorporated in the TMT	27
3.4.1. Topic visualization and annotation	29
3.4.2. Topic curation tools.....	32
3.5. Evaluation of topic models	34
4. Software Documentation.....	36
4.1. Software architecture	36
4.2. Requirements	40
4.3. Configuration file	41
4.4. Topic Modeling project description	46
4.4.1. Input folders	46
4.4.2. Description of entities.....	48
4.5. Command line API for Backend services support.....	53
4.5.1. Corpus Management services (src/ManageCorpus).....	55
4.5.2. List Management services (src/ManageLists)	56
4.5.3. Topic Modeling services (src/topicmodeling)	57
4.5.4. Deployment flow	60
4.6. TMT Front Ends.....	61
4.6.1. Execution commands	62
4.6.2. Command line user interface	63

4.6.3. Graphical user interface	63
5. Performance Evaluation.....	69
5.1. Data Preprocessing	70
5.1.1. Parallelization by employing the Dask framework	70
5.1.2. Parallelization by employing the Spark Framework	75
5.2. Topic Modeling	76
6. Conclusions and Future Work.....	82
References.....	84

FIGURES

Figure 1. Workflow for the training and curation of a topic model	16
Figure 2. Illustration of the operation of the components in the TMT preprocessing pipeline..	19
Figure 3. High-level sketch of ProLDA.....	23
Figure 4. High-level sketch of Contextualized Topic Models.....	25
Figure 5. Graphical representation of HTM-WS and HTM-DS corpus generation for fictional topic 2 (represented in blue)	27
Figure 6. Topics description for a topic model of 60 topics of S2CS (Computer Science papers in Semantic Scholar). The first 13 topics are shown.	29
Figure 7. pyLDAvis output for a topic model of 60 topics of S2CS (Computer Science papers in Semantic Scholar). Topic 32 (NLP) has been selected.	32
Figure 8. Suggestion of most similar pairs of topics for a topic model of 60 topics of S2CS (Computer Science papers in Semantic Scholar).	33
Figure 9. Snapshot of the software documentation	38
Figure 10. Graphical user interface; The welcome page	62
Figure 11. Command line user interface main menu	63
Figure 12. Available local datasets view	64
Figure 13. Training dataset generation window	64
Figure 14. Available training datasets view	65
Figure 15. Topic modeling preprocessing window	66
Figure 16. Topic modeling training window	66
Figure 17. Wordlists management subwindow	67
Figure 18. Wordlists creation (left) / edition (right) subwindows	67
Figure 19. Model management subwindow	68
Figure 20. Curation subwindow	69
Figure 21. Settings subwindow	69
Figure 22. Cleaning and export times for the CORDIS dataset using Dask with a variable number of workers	71
Figure 23. Memory and CPU usage for preprocessing and corpus preparation tasks using Dask with a variable number of workers. Dataset S2CS sampled at 10% (1,480,804 documents)	72
Figure 24. Memory and CPU usage for preprocessing and corpus preparation tasks using Dask with a variable number of workers. Dataset S2CS (14,801,878 documents)	73
Figure 25. Time for generating the Gensim dictionary for the S2CS dataset as a function of the number of documents (varying sampling rate).....	74
Figure 26. Total time for the S2CS dataset preprocessing (left) and total time just for the parallelized tasks as a function of the number of documents (varying sampling rate)	74
Figure 27. Total time for S2CS dataset preprocessing using the Spark based implementation running over a Spark cluster with 10 nodes, using a varying number of cores per node	76
Figure 28. Topic model comparison in terms of c_nmpi coherence for a variety of training parameters for Mallet, ProLDA, and CTM implementations	78
Figure 29. Topic model comparison in terms of c_nmpi coherence for a variety of training parameters for Mallet, ProLDA, and CTM implementations	79

Figure 30. Topic model comparison in terms of u_mass coherence for a variety of training parameters for Mallet, ProdLDA, and CTM implementations80

Figure 31. Topic model comparison in terms of u_mass coherence for a variety of training parameters for Mallet, ProdLDA, and CTM implementations.81

TABLES

Table 1. Document revision history 2

Table 2. Most relevant setting parameters for Mallet Topic Modeling 20

Table 3. Most relevant setting parameters for Spark MLLIB Topic Modeling 21

Table 4. Most relevant setting parameters for AVITM Topic Modeling 24

Table 5. Most relevant setting parameters for Contextualized Topic Modeling 26

Table 6. Setting parameters for the construction of level-2 topics models 27

Table 7. Most relevant coherence metrics in the literature 35

Table 8. List of libraries used by the Topic Modeling Toolbox. For each required library we indicate the version that has been used for the current release and their licenses.. 40

Table 9. Fields of the TMT entity “Local dataset” 49

Table 10. Fields of the TMT entity “Training dataset” 50

Table 11. Fields of the TMT entity “Wordlist” 51

Table 12. Fields of the TMT entity “Topic Model” 53

ACRONYMS

AI — Artificial Intelligence

ARC — Athena Research Centre

BOW — Bag of Words

BTM — Bayesian-based topic models

CTM — Contextualized Topic Modeling

GUI — Graphical user interface

HTM — Hierarchical Topic Model

HTM-DS — Hierarchical Topic Model with Document Selection

HTM-WS — Hierarchical Topic Model with Words Selection

IMT — Interactive Model Trainer

LDA — Latent Dirichlet Allocation

LL — Living Labs

NLP — Natural Language Processing

NTM — Neural topic models

STI — Science, Technology and Innovation

TMT — Topic Modeling Toolbox

UC3M — University Carlos III of Madrid

UI — User interface

EXECUTIVE SUMMARY

IntelComp is a project funded by the European Commission whose main objective is the development of a software platform that, using the latest generation of Artificial Intelligence (AI) and Natural Language Processing (NLP) tools, provides relevant information for the management of public policies in the area of Science, Technology and Innovation (STI), to help decision making along the whole policy cycle. To do this, the platform will analyze various documentary sources (e.g., scientific production documents, research call work programs, information on companies available on the Internet, etc.), some of them involving hundreds of millions of documents, and, through the application of AI techniques and models, it will extract information with a level of detail greater than that available in the metadata that usually accompanies these data sources.

One of the main tools in IntelComp is topic modeling, whose objective is to carry out an unsupervised analysis of any set of documents, extracting from a direct analysis of the text the main themes the collection of documents deals with, and the relation of each analyzed document with the identified topics. Therefore, the main output of Task 3.5 is a python-based toolbox (hereinafter “TMT”) which is available at <https://github.com/IntelCompH2020/topicmodeler>). This document provides information about said toolbox, including a review of the techniques implemented, the description of the architecture of the software, and manuals for the users of the tool.

Some of the principles that have guided the design of the toolbox are:

- inclusion of different algorithmic solutions for the preprocessing of the texts and for the topic modeling itself, in order to be able to select the most suitable implementation at any time according to the characteristics of the corpus to be analyzed, the scalability characteristics of the tools, and the infrastructure available for their execution.
- incorporation of expert knowledge during the training of the topic models in order to take advantage of their previous experience and obtain models that are more interpretable by the final users of IntelComp (expert-in-the-loop approach).
- dockerization of the components to facilitate the future integration of the TMT services in IntelComp end-user applications such as the Interactive Model Trainer (hereinafter “IMT”). To this end, the implementation of the toolbox has followed the design principles provided by WP5.
- development of two user interfaces in Python, not conceived for use in IntelComp, but to make the toolbox itself a completely autonomous software that can be published in the open.

Another output of Task 3.5. is the generation of topic models for the different domains and data sources considered in IntelComp. These models will be trained immediately following the release of the TMT, and especially after M21 when the IMT will be available to all Living Lab (LL) users. The development team of Task 3.5. will then work along with the LL users to provide the

necessary training to use the topic modeling functionalities in the IMT, and to adopt any necessary modifications to improve user experience.

Taking into consideration all the previous, this document provides:

- Technical information about the developed software that can be used by other implementation teams of IntelComp, particularly addressed to the team responsible for integration and implementation of the front end of the Interactive Model Trainer
- A complete guide of use of the current toolbox release, as well as a number of practical advices, that is aimed to LLMs users, to provide them with a minimum background information about topic modeling, and how to use the toolbox functionalities to obtain high-quality topic models
- Information for a general AI practitioner that may find convenient the use of the toolbox for his/her own projects.

1. INTRODUCTION

IntelComp is a project funded by the European Commission whose main objective is the development of a software platform that, using the latest generation of Artificial Intelligence (AI) and Natural Language Processing (NLP) tools, provides relevant information for the management of public policies in the area of Science, Technology and Innovation (STI). To do this, the platform will analyze various documentary sources (e.g., scientific production documents, research call work programs, information on companies available on the Internet, etc.), some of them involving hundreds of millions of documents, and, through the application of AI techniques and models, it will extract information with a level of detail greater than that available in the metadata that usually accompanies these data sources. Likewise, the extracted information will be analyzed jointly with other available metadata through advanced visualization techniques included in the various user tools that will be developed in the project.

The design of automatic analysis tools for those sources must take into account two fundamental aspects of great relevance for IntelComp: 1) the sources to be analyzed have a high degree of heterogeneity, in the sense that any given data set can deal with very diverse topics, but also in terms of the difference in vocabularies used by the various textual sources analyzed (e.g., scientific articles vs job offers); and 2) the direct use of AI tools without any type of expert validation may produce results that can be difficult to interpret by the end users, which makes it advisable to design procedures that allow domain experts to be involved and engaged in the creation and curation of the developed models (i.e., following an expert-in-the-loop approach) [1].

One of the main tools in IntelComp to address the two mentioned challenges is topic modeling [2]. The objective of this NLP technology is to carry out an unsupervised analysis of any set of documents, extracting from a direct analysis of the text the main themes the collection of documents deals with, and the relation of each analyzed document with the identified topics. The main advantage of topic modeling compared to other available AI/ML technologies is that its application does not require any labels associated with the documents. In addition, the widespread adoption of this technology in various fields of application, including studies for STI policies [3], [4], [5], [6], [7], suggests that the results obtained with topic modeling provide relevant information for STI policy-makers.

The use of topic modeling tools in IntelComp has a dual purpose: 1) to provide a greater level of detail of the data sources used in IntelComp than that available from the originally available metadata, and 2) to provide a space of document representation on which other analysis tools can be built, such as impact measurement from semantic similarity graphs (considered in Task 3.7 “Impact and influence using graph analysis”). In this sense, the space provided by topic modeling can be used to develop metrics that are aligned with the level of semantic similarity among documents [1].

The main output of the topic modeling task (Task 3.5) is the python-based Topic Modeling Toolbox (TMT) toolbox available in an open Github repository¹. This document provides

¹ <https://github.com/IntelCompH2020/topicmodeler>

information about the TMT toolbox, including a review of the techniques implemented, the description of the architecture of the software, and manuals for the users of the tool. It should be mentioned that the design of the application has taken into account certain integration principles agreed within the project, which will allow an easier integration with the Interactive Model Trainer (IMT), one of the end users IntelComp applications.

Regarding the algorithmic developments that have been carried out within the task, the software includes different solutions for modeling topics, with the aim of being able to validate which of them provides a better tradeoff between performance and computational requirements, also considering the various execution scenarios that Barcelona SuperComputing has provided for the deployment of IntelComp. In addition, a series of proprietary algorithmic improvements have been developed that allow topic models to be curated by domain experts. This set of tools is a distinctive feature of topic modeling in IntelComp as it provides an expert-in-the-loop approach that will ultimately contribute to models that are more aligned with the prior experience and needs of IntelComp end users.

The developed TMT includes two different front-ends for building and curating topic models. Both front-ends are developed entirely in python and allow interaction either through a set of menus in a command terminal, or through a graphical user interface. Internally, both visualizations are decoupled from the actual training and optimization of the models, which will facilitate the development of a front-end based on a web service, which will be the option used for the IMT, and which is already being developed as part of IntelComp's Task 4.2. "Interactive Model Trainer".

Another output of Task 3.5. is the generation of topic models for the different domains and data sources considered in IntelComp. These models will be trained immediately following the release of the TMT, and especially after M21 when the IMT will be available to all Living Lab (LL) users. The development team of Task 3.5. will then work along with the LL users to provide the necessary training to use the topic modeling functionalities in the IMT, and to adopt any necessary modifications to improve user experience. Other tasks that will take place are the dockerization and deployment of services together with Task 5.4 (integration), although the development of the toolbox already considered the future needs of the IMT, as we have already discussed.

The rest of the document is structured as follows. In the next section, we present the specific objectives of the topic modeling task, which justify the decisions taken during the design of the TMT, both in terms of the selection of topic training and curation tools, as well as those related to the architecture of the software application. Next, in Section 3, the theoretical principles of probabilistic topic modeling are concisely reviewed, and the specific algorithms that have been included within the toolbox are described. Section 4 describes the TMT in detail, providing all the necessary documentation for its deployment and use. The python tools developed for the direct use of the application (terminal-based UI and Graphical User Interface (GUI)) are also described. The document ends with a comparative evaluation of the techniques included in the software and the corresponding conclusions and lines of improvement that could be addressed.

2. OBJECTIVES

The main objective of Task 3.5 of the IntelComp project is the design and implementation of a topic extraction toolbox that can serve the specific requirements of the project, specifically in terms of performance, scalability, incorporation of mechanisms to incorporate the knowledge of the domain experts (i.e., an expert-in-the-loop approach), and integration with other WP3 components and the end-user IntelComp tools. This general objective can be broken down into the following specific objectives that have guided the execution of the work:

- *Incorporation and testing of different topic extraction libraries*, considering both classic techniques based on Variational Bayes, as well as more recent ones based on neural networks, including implementations that incorporate advanced language models (transformers). Specifically, the optimization of the models using Mallet, Spark LDA, proLDA and Contextualized Topic Modeling (CTM) have been incorporated into the TMT, evaluating these algorithms in terms of the coherence of the extracted topics and their scalability.
- *Definition of a general paradigm for the representation of topic models*, regardless of the method used for training. In this way, it is intended to provide a general framework on which curation tools can be used, regardless of the particularities of the topic modeling libraries used to build the model.
- *Exploiting this common representation*. It is desired to provide a series of topic curation tools to better align the obtained models with the knowledge and previous experience of domain experts. These tools will allow performing tasks such as the elimination of noisy topics, merge of similar topics, etc., also providing relevant information to carry out these actions based on the statistical properties of the topics (e.g., their composition or their presence on the documents of the corpus).
- *Provide useful tools and visualizations for the domain experts who are going to carry out the curation of the models*. Those visualizations will serve as a first test battery for the implementation of the final tools that will be available in the IMT, and will allow training the first topic models until the IMT is available for general use within the project. In any case, the implemented user interfaces will add value to the IntelComp TMT as a self-contained toolbox that will be released as an open-source project result.
- *Provide a novel implementation of Hierarchical Topic Models (HTMs)*. We will provide a 2-level HTM that incorporates tools to allow the user to pick which topics should be further split. Again, in addition to the implementation of the hierarchical modeling method itself, it is necessary to extract a series of statistics that offer the user information about which topics are likely good candidates to be split, as well as the goodness of the nested models that are obtained.
- *Serve as a testbench for the integration of machine learning tools in the IMT*, using the philosophy of providing the services for training, curating, etc. as docker-exec available services. These general principles have been established in the IntelComp WP5 and allow decoupling the design of AI services from the implementation of user tools, as well

as facilitating the deployment of each component in the most suitable infrastructure for its execution.

3. TOPIC MODELING

The growing interest in methods based on NLP and machine learning has driven intense research work for its application in the field of STI analysis. Topic modeling aims at analyzing a corpus of text documents, and automatically identifying and characterizing the most relevant topics appearing in the document collection. Not requiring any previous step in terms of manual labeling of the documents that make up the corpus, it is therefore not surprising that topic modeling has been widely used in the field of STI analysis, for example to analyze scientific production, patents, or research projects, using a variety of corpora of open documents. As an example, the proceedings of the latest editions of the Global Tech Mining Conference² include numerous contributions based on topic modeling, and even dedicated sessions in which the advantages and disadvantages of this technique for STI analysis are evaluated, offering some good practices or algorithmic modifications to improve its characteristics in this field of application.

Although there are some precursor algorithms, we can stipulate that Latent Dirichlet Allocation (LDA) proposed by David Blei in 2003 [2] constitutes a turning point in terms of the popularity and extensive use of topic modeling tools. So much so that LDA has been applied in a wide variety of application areas, and has given rise to a large number of tools and algorithmic variants, for example for the temporal analysis of topics [8], the construction of hierarchical models [9], or incorporating correlation between topics [10], etc. Recently, with the rise in popularity of neural networks and language models based on attentional models (transformers), some topic models that make use of these AI tools have been published [11], [12], [13], [14].

In order to be able to evaluate the characteristics of the different available libraries, this project incorporates some of the most popular and widely used classic tools such as Mallet and Spark LDA, both based on the generative model originally proposed by David Blei, as well as others more experimental but that promise to be widely adopted in the near future, such as prodLDA and, above all, CTM, an extension of ProdLDA that incorporates attentional models based on Transformers. All these alternatives are incorporated into the development carried out in Task 3.5, and have been unified into a Python class that offers a level of abstraction that allows the exploitation of any of them with a set of tools for topic curation.

With respect to the widespread use of statistical approaches supported by taxonomies, LDA has some properties that are useful for STI analysis. Among them are:

- Thematic analysis can be carried out with different levels of resolution, or hierarchically. This helps in the survey of specific subject areas with the desired level of detail.

² <http://www.gtmconference.org/>

- Its flexibility allows the identification of emerging topics, or the detection of hybridization of topics, something hard to do using taxonomies that are often exclusive and whose updating imposes certain time delays.
- Since LDA also provides a vector representation of documents, it is possible to carry out a semantic comparison between documents of different data collections, even in the absence of a common taxonomic representation. Also, the time shift of certain topics in one corpus with respect to another can be studied (lead-lag analysis).

In IntelComp, the use of topic modeling is considered both as a tool to analyze a certain data set of documents with a great variety of topics, and to carry out an analysis with a high level of granularity of a specific area, allowing to discover more detailed topics than those available in a given taxonomic classification. In any case, the objective of Task 3.5 is to provide a toolbox that can be used to train topic models on any collection provided by the user, as well as to curate the topics obtained so that the model is more interpretable by the end users who will use it, for example, to analyze the relative size of the identified topics, their evolution over time, or the strengths or weaknesses of specific geographical areas in terms of their contribution to a specific topic.

For the purposes of a general understanding of the operation of any of the topic modeling tools incorporated into the IntelComp Topic Modeling Toolbox, and avoiding an overly technical description for which the interested reader can consult a large number of scientific papers³, in this document we will adopt a black-box approach of all the topic extraction techniques, whose operation can be described in a general way as a block which takes as input the texts that make up the collection of documents, and provides the following two outputs:

- A list of predominant topics, where each topic is defined by a weighted list of characteristic terms.
- A mapping of documents from the word space into a topic space, in which each document is characterized by a vector of length equal to the number of topics, each component yielding the proportion of the document that is assigned to its corresponding topic.

The different tools used may differ in the way in which these outputs are obtained from the inputs, and their detailed description is outside the scope of this document, but our black box treatment is sufficient to understand that it is possible to provide a common abstraction that allows representing the topic models that are trained by the tool, regardless of the specific library used for their training.

In addition to the previous description, in order to understand the operation of topic analysis tools, it must also be considered that these tools are usually based on a representation known as a “bag of words” of the corpus documents, which consists of a simple count of the number of occurrences of each word within each document. In order to obtain a more effective representation, it is essential to eliminate common terms or terms with little semantic

³ E.g., we recommend the comprehensive work of [31] that provides a good balance between simplicity of the exposition and technical rigor.

importance from the vocabulary, as well as to unify morphological variants of the same word. To obtain this representation, IntelComp relies on the NLP pipelines implemented in Task 3.1, and specifically on the lemmatization and N-gram identification components.

Finally, it should be mentioned that in addition to building topic models, it is essential to provide the tools that allow inference of the main topics in documents that were not present in the training set. For this, it is essential to replicate during the inference process both the analysis corresponding to the selected topic model, as well as the previous preprocessing of the text.

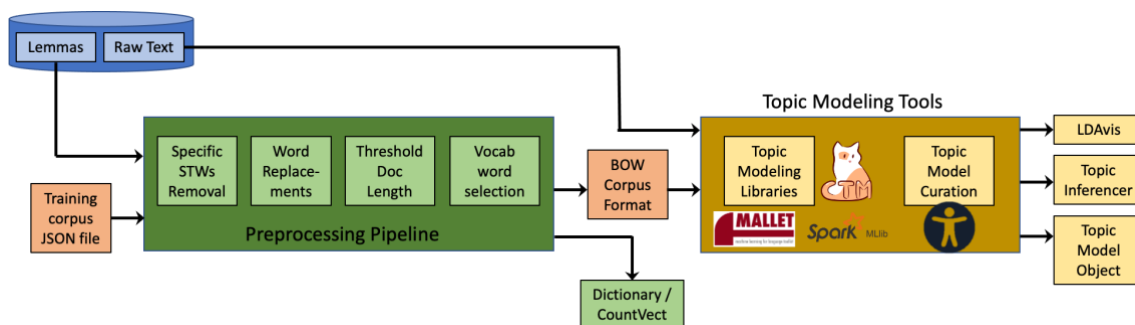
The next subsection presents a more detailed description of the pipelines implemented in the toolbox for topic modeling training and curation, as well as for inference on new documents.

After that, we will turn our attention to the specific procedures that implement each of the identified functional blocks.

3.1. Description of the Pipeline supported by the toolbox

The construction of a topic model is a procedure that requires the sequential execution of various tasks. Figure 1 shows a block diagram that represents the general implementation of the procedure in the TMT.

Figure 1. Workflow for the training and curation of a topic model



The input to the process is a corpus for training the model, for which the toolbox assumes the availability of a series of data sets in parquet format, including both the raw text and lemmatized versions of the texts that are obtained as output from the NLP pipeline provided by Task T3.1. For the construction of a training corpus for a specific model, the implemented SW allows the concatenation of documents coming from several data sets. The information necessary for the construction of the training corpus from the original data sets available on the platform is stored in a JSON file, so that each training corpus is associated with a different JSON file. The TMT includes among its tools a series of functions to create, list or delete training corpus for topic modeling.

Once a well-defined training set is available, the next step to be carried out consists of cleaning and homogenizing the text of the training data. To this end, the TMT provides a pre-processing pipeline that executes a series of basic tasks and that can be adjusted to each training data corpus, also incorporating information provided by the experts who are building the model (e.g., stopwords, equivalent terms or acronyms for a particular model, etc). It is important to highlight that the pre-processing pipeline incorporated in the TMT is not a complete NLP pipeline, but only provides additional cleansing and normalization tasks that are usually recommended to be

carried out in order to obtain higher quality topic models with more easily interpretable descriptions. The TMT assumes that heavy preprocessing, including lemmatization and N-gram extraction or entity detection, have been performed previously so that such procedures have already been performed on the text before the TMT preprocessing pipeline. The output of the preprocessing pipeline will be a training corpus prepared for the subsequent training of the topic model, in which each text has been represented as a bag of words (BOW), as well as the vocabulary of terms that have been selected for such BOW representations.

Finally, the training of the topic model itself must be carried out. Depending on the technique used, only the BOW representation of the texts, the unprocessed raw text, or both representations can be used. For instance, when Mallet is used only the BOW representation is required, whereas CTM requires raw text in addition to BOW. As a result of this training, and regardless of the technique used, a common representation of the model is generated, on which users can perform certain curation tasks, for example, tagging topics, deleting noisy topics, etc. The result is a model ready for deployment within the platform.

As already mentioned, it is important to note that the trained model will be exploited by the platform to analyze its composition in terms of the nature of the topics found or its size over time, but also to carry out inference tasks on new unseen texts not available during training. Such inference will require replicating the processing pipelines used during model training and subsequently applying the model inference tool along with the necessary transformations associated with expert curation activity. Therefore, during the construction of the model and regardless of the tools used, it must be guaranteed that the information necessary for the inference of topics is preserved.

In the rest of this section, we analyze in detail the different components that have been included in the TMT from a functional point of view, i.e., describing its operation without going into excessive detail about aspects associated with the actual SW implementation. These implementation-related aspects will be addressed in Section 4.

3.2. Description of the Pipeline supported by the toolbox

This subsection explains the different tasks that the TMT carries out to build a corpus that allows the training of a topic model. The TMT assumes that the text of the documents is available in parquet format within a folder accessible by it. It is also considered that the text is already pre-processed so that, together with the raw text, a lemmatized version and/or extraction of named entities (or n-grams) is available. For the above, the project SW repository includes several auxiliary Jupyter Notebooks, which allow basic lemma extraction using the Spark NLP lemmatizer⁴.

The construction of the training dataset has as its first step the concatenation of the documents that make up the dataset. The provided toolbox includes a training corpus management module, which allows documents from different data sets to be incorporated into the same training set.

⁴ <https://nlp.johnsnowlabs.com/docs/en/annotators#lemmatizer>

This collection of documents where each document is represented as a list of lemmas, is then used as input to the preprocessing pipeline consisting of the following steps:

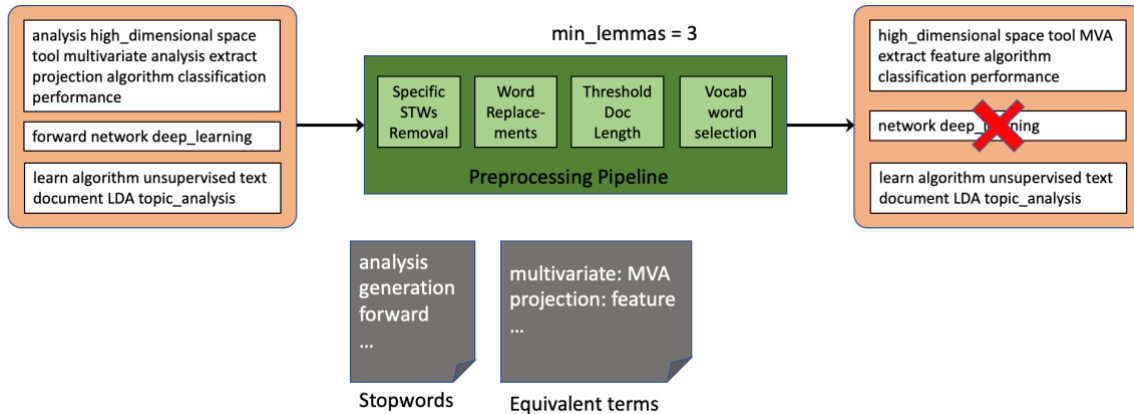
1. **Stopwords removal:** lemmas that are included in a list of stopwords are eliminated from the set of lemmas of each document. It is important to mention that generic stopwords will have already been previously filtered from the input data, and that the objective of this new step of eliminating stopwords is to eliminate those words that, although cannot be considered as general stopwords for the language used, lack semantic interest for the specific dataset analyzed. As an example, "computer" cannot be considered a general stopword for the English language, but it is probably best to remove it when analyzing a data set restricted to the computer science domain.
2. **Word replacements:** it consists of the replacement of certain lemmas for other equivalent ones. In this way, words with equivalent semantic meaning, for example acronyms, can be mapped to the same terms and treated as a single term during topic modeling. Likewise, this component can be used to solve some errors in the lemmatization block that will be evident in the visual description of the topics, downgrading the apparent quality of the resulting topic model.
3. **Filtering of short documents:** those documents that do not have a minimum number of lemmas are eliminated from the training set, considering that there is not enough information available for a robust estimation of the thematic composition of the document.
4. **Vocabulary construction:** Particularly important in very large training sets, it is convenient to limit the number of terms that make up the vocabulary. Three different procedures are applied for this. In the first place, we remove from the vocabulary terms that do not appear in a sufficient number of documents, since on a good number of occasions they can be typos or, in any case, cannot be reliably assigned to any topic. Secondly, those terms that appear in a number of documents greater than an established threshold will also be filtered out, considering that they are words of little semantic interest for the analyzed corpus. Finally, for practical reasons of scalability of the topic extraction tools, the maximum size of the vocabulary is restricted, keeping only the most frequent terms that have passed the two previous selection criteria.

For the generation of lists of stopwords and equivalent terms, the TMT includes a term list management component that can be used to create additional lists from the results of a given model. Preprocessing allows one or more of such lists to be selected from among the available ones, both for the selection of stopwords and for the equivalent terms.

Once the vocabulary is available, the BOW representation of each document can be calculated by simply counting the number of occurrences of each vocabulary term in each document. The output of the processing pipeline is therefore a subset of the input documents (those with sufficient length) each represented by a vector of length equal to the number of terms in the vocabulary, known as a bag of words (BOW) representation, where any type of structure of the original document has been discarded and only the information related to the number of appearances of each term is preserved. Since a small number of vocabulary words will usually

appear in each document, these vectors are usually efficiently represented as sparse vectors in which only the non-zero components and their position indices are preserved. The complete text preprocessing procedure has been illustrated in Figure 2 for a concrete example.

Figure 2. Illustration of the operation of the components in the TMT preprocessing pipeline



The tool included in the TMT includes two different implementations that allow highly parallelizable executions of the described preprocessing procedures.

- The first implementation allows parallelizing document transformations using Spark, and is considered especially useful since IntelComp has a Spark-based computing cluster for deploying the platform with very large horizontal scalability capabilities. In this case, the execution of all the components of the processing pipeline can be easily distributed among the nodes of the cluster, including the calculation of the vocabulary that is performed with the `CountVectorizer` Spark object.
- For those cases where deployment to a Spark cluster is not available, we have implemented a version of the pipeline based on Dask dataframes⁵ which is a transformation-based preprocessing pipeline enabling parallelization of the operations to be carried out on the documents, thus, catering for a better use of computational resources and more efficient memory management. This implementation allows you to take advantage of the available resources when the container is deployed on a virtual machine. However, the vocabulary construction is done using Gensim dictionaries, which requires sequential processing of the documents, this procedure being considerably slower than the corresponding implementation in Spark.

3.3. Topic Modeling technologies incorporated in the TMT

There are different techniques available for extracting the most important topics from a corpus of text documents. All the algorithms used within the TMT have in common an output format with two output matrices obtained:

- A topic composition matrix (β): it is a matrix of size (number_of_topics \times vocabulary_size), in which each row contains the probability distribution that

⁵ <https://www.dask.org/>. The concept of a Dataframe in Dask is similar to Pandas, but Dask allows dataframe partitioning and parallel calculations.

characterizes the frequency of appearance of the different terms of the vocabulary for each identified topic. Therefore, all elements in matrix β are non-negative values, and the sum of the elements along each row is one. From this matrix, the nature of each topic can be explored from the words that accumulate the highest probability.

- Matrix of thematic composition of the documents (θ): it is a matrix of size (number_of_documents \times number_of_topics), in which each row contains the proportion of the topics identified for the corresponding document. Again, matrix θ contains non-negative values, and the sum of the elements in each of its rows is one. Matrix θ allows, therefore, to analyze the most relevant topics for each of the documents in the corpus. We refer to the set of topics whose associated weight is larger than zero for a document as the *active topics* of that document.

Although the output can be described in a common way, a variety of tools that differ in their internal characteristics and scalability properties have been incorporated into the toolbox. Next, we review the different topic extraction libraries that have been included.

3.3.1. Mallet

IntelComp allows the training of topic models using Mallet, a very popular library written in Java that presents a highly efficient parallelizable implementation of LDA. Mallet has been extensively used by the UC3M and ARC teams in previous projects involving data sources of similar nature to the ones used in IntelComp obtaining very satisfactory results. Likewise, Mallet has also been exploited in many other works that have appeared in the technical literature over the last years. In recent years, other python libraries for LDA have become very popular, such as the Gensim and scikit-learn implementations but, in our experience, they offer worse performance in terms of topic coherence and scalability than Mallet. Therefore, we stick with Mallet as our preferred choice for deployment over machines with sufficient computation and memory resources.

Mallet performs model optimization with Gibbs Sampling, a Markov Chain Monte Carlo (MCMC) technique [15] that is well suited for parallel optimization. Multithreading allows Mallet to make good use of the available resources leading to fast training. The following list summarizes some of the most relevant configuration parameters available for Mallet training. Default settings are provided for all of them in the TMT and can be modified in the configuration file of the training process. Additionally, the advanced training tools available through both the command line and graphical UIs allow users to modify these default settings for each specific training.

Table 2. Most relevant setting parameters for Mallet Topic Modeling

Parameter	Default	Description
num_topics	25	Number of topics for the model. A large number of topics will provide a higher level of detail, and will allow us to find some very specialized topics of small size. One drawback, however, is that larger topics can appear repetitive and fragmented. On the contrary, a small number of topics is useful to have a first approximation of the main topics of the corpus.
alpha	5	Hyperparameter for the Dirichlet distribution responsible for generating the topic vectors of the documents. A small value favors

		having few active topics per document; on the contrary, a high value will generally imply a greater number of active topics.
num-iterations	1000	Number of Gibbs sampling iterations. A common setting is 1000, although in order to have a rough first model it may be enough to use values around 250.
optimize-interval	10	Number of Gibbs sampling iterations between reestimates of the hyperparameters of the probability distributions. If the parameter is set to 0, it implies that the initial values will be preserved, and hyperparameters will not be optimized during model training.
doc-topics-threshold	0	Threshold for pruning topics in the document representation matrix.
num-threads	4	Number of threads for the parallel optimization of the algorithm.

3.3.2. Spark MLLIB

Spark incorporates a machine learning library, MLLIB, which includes two LDA implementations among the available algorithms. The first one is an online implementation of David Blei's original algorithm based on Variational Bayes, while the second one is based on an expectation-maximization algorithm (i.e., two steps are iteratively executed: the calculation of the mathematical expectations of a set of latent variables and their maximization). IntelComp offers among the available platforms for the deployment of the components a Spark cluster that provides a great capacity for the horizontal scaling of the algorithms. For this reason, the inclusion of this implementation of topic modeling in the catalog of tools available in the TMT has been considered of great interest.

The LDA implementation of Spark includes a series of parameters that can be modified to adjust the behavior of the algorithm. As in the case of Mallet, the toolbox allows configuring default values for some of these parameters, as shown below in Table 3. As in Mallet, of all the available parameters, the TMT allows modifying only those that have a clearer meaning for the user, such as the main optimization parameters (e.g., a greater number of iterations offers greater precision at the cost of increased computation time) or those that have a clear impact on the results (such as the number of topics or the prior of document distribution, which directly influences the average number of active topics). In the case of Spark MLLIB, we deliberately avoid offering advanced options for the adjustment of other available parameters that require greater expertise from the users, such as the adaptation steps for online learning, or the concentration parameter for the word-topic distribution.

Table 3. Most relevant setting parameters for Spark MLLIB Topic Modeling

Parameter	Default	Description
num_topics	25	Number of topics for the model. A large number of topics will provide a higher level of detail, and will allow us to find some very specialized topics of small size. One drawback, however, is that larger topics can appear repetitive and fragmented. On the contrary, a small number of topics is useful to have a first approximation of the main topics of the corpus. This value is commonly referred to as parameter 'k' in Spark MLLIB documentation.

alpha	5	Hyperparameter for the Dirichlet distribution responsible for generating the topic vectors of the documents. A small value favors having few active topics per document; on the contrary, a high value will generally imply a greater number of active topics. This value is commonly referred to as parameter 'docConcentration' in Spark MLLIB documentation.
maxIterations	20	Maximum number of iterations for the optimization algorithm
optimizer	'online'	The algorithm used for algorithm optimization. Two optimizers are available in Spark MLLIB LDA: Expectation-maximization ('em') and online variational Bayes Optimization ('online').
optimizeDocConcentration	True	Whether the doc-concentration parameter alpha will be optimized during the training of the algorithm.
subsamplingRate	0.05	Percentage of documents from the training set that will be used at every minibatch.

3.3.3. Neural Topic Models

Bayesian-based topic models (BTMs), with Latent Dirichlet Allocation (LDA) as a representative, have been a powerful technique for text analysis in many fields for almost two decades. Yet, in today's era headed by unprecedented interest in AI, conventional BTMs have started to wear out. In this regard, we can cite three main weaknesses of BTMs: 1) Their inference process (carried out through a Bayesian inference process, e.g., variational inference of Monte Carlo sampling methods) needs to be usually ad-hoc customized and the inference complexity may grow with the model difficulty. 2) The inference process may be hard to scale, especially with collections of documents; additionally, these processes are not easily leveraged through parallel computing facilities like GPUs. 3) It may be useful to benefit from the joint combination of topic models and deep-structured neural networks, in particular specific neural structures such as BERT and Transformers, that allow to boost the topic model learning process incorporating language information gathered from processing huge data sets.

Hence, aiming to boost performance and provide conventional topic models with flexibility and scalability capabilities, neural topic models (NTMs) have gained a huge research interest, and many variants are currently in development. This has led to the application of NTMs to important tasks in the field of NLP, including text generation, and document summarization, which are areas in which conventional topic models are not straightforward to apply.

One current appealing line of research in the field of NTMs is the incorporation of pre-trained language models as a means of endowing topic models with more advanced and finer-grained abilities to capture aspects of linguistic context through contextual word- and sentence-vector representations, see e.g., *BERTopic* [14], *TopicBERT* [16] or *Neural Variational Document Model (NVDM)* [17], just to name a few examples. Though most of these proposals are still in a preliminary stage, the project has chosen to incorporate two of the most representative NMT techniques to evaluate their performance in comparison with the classical approaches described in Subsections 3.3.1 and 3.3.2. In particular, the NTMs incorporated into the TMT are the following ones:

Product-of-Experts LDA

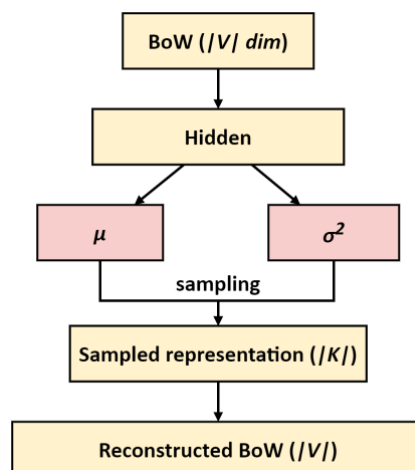
ProdLDA (Product-of-Experts LDA) [11] is a state-of-the-art neural topic model that utilizes Autoencoded Variational Inference for Topic Models (AVITM), a black-box variational inference method [18] that can be easily applied to new models and reduce the computational cost associated with the posterior distribution's computation in conventional topic models.

AVITM has its basis on mean-field variational inference, but in contrast to the traditional mean-field approach, the variational parameters are computed through a neural network that takes the observed data w (the documents) in BOW format as input and maps them into a continuous latent representation. Then, a second neural network (decoder) is in charge of performing the inference process, i.e., of reconstructing the BOW by generating its words from the latent document representation.

With respect to the AVITM based implementation of LDA, the authors constructed a Laplace approximation to the Dirichlet prior utilizing Gaussian distributions. Then, ProdLDA can be described as a modification of the AVITM-based implementation of LDA, in which the word-level mixture over topics is carried out in a natural parameters space, i.e., the topic matrix is not constrained to exist in a multinomial simplex prior to mixing. That is, the only difference between LDA AVITM-based and ProdLDA is that in the latter, a weighted product-of-experts [19] replaces the multinomial distribution over individual words.

The network structure of prodLDA has been represented in Figure 3. Each document BOW representation (of dimension equal to the vocabulary size) is processed by a set of hidden layers that learn the mean and variance of a Gaussian distribution. Based on these parameters the document representation is obtained as a set of K values from which the network tries to recover the initial document representation. The topic modeling inference consists of learning the network parameters to optimize the reconstruction, and document topic-based representations and topic characterizations are obtained as a subproduct of this process.

Figure 3. High-level sketch of ProdLDA



The claimed advantages of ProdLDA include 1) better topics than LDA in terms of coherence; 2) training is fast and efficient like standard mean-field, and when trained on new data, much faster as only one pass through the neural network is required.

The TMT toolbox allows both the training of the AVITM-based implementation of LDA and ProLDA; Table 4 summarizes their most relevant configuration parameters. These parameters can be configured by the user, in addition to some other advanced settings that are only shown if the user requests them

Table 4. Most relevant setting parameters for AVITM Topic Modeling

Parameter	Default	Description
num_topics	25	Number of topics for the model. A large number of topics will provide a higher level of detail, and will allow us to find some very specialized topics of small size. One drawback, however, is that larger topics can appear repetitive and fragmented. On the contrary, a small number of topics is useful to have a first approximation of the main topics of the corpus. This value is commonly referred to as ‘number of components’ in the AVITM implementation.
model_type	ProdLDA	The AVITM implementation used for training, namely ProdLDA or LDA.
num_epochs	100	Number of complete passes through the training dataset over which the model will be trained. It can be set to an integer number between zero and infinity
batch_size	64	Number of samples in the minibatch for neural network training. It should have a value larger or equal to one and less or equal than the number of samples in the training dataset.

Contextualized Topic Models

Contextualized Topic Models [12], [13] consist of a family of neural network-based topic modeling algorithms. Currently, four different versions of CTMs have been proposed by the authors, namely CombinedTM, ZeroShotTM, Supervised CTM, and β -CTM, all of which are constructed by extending AVITM under different conditions.

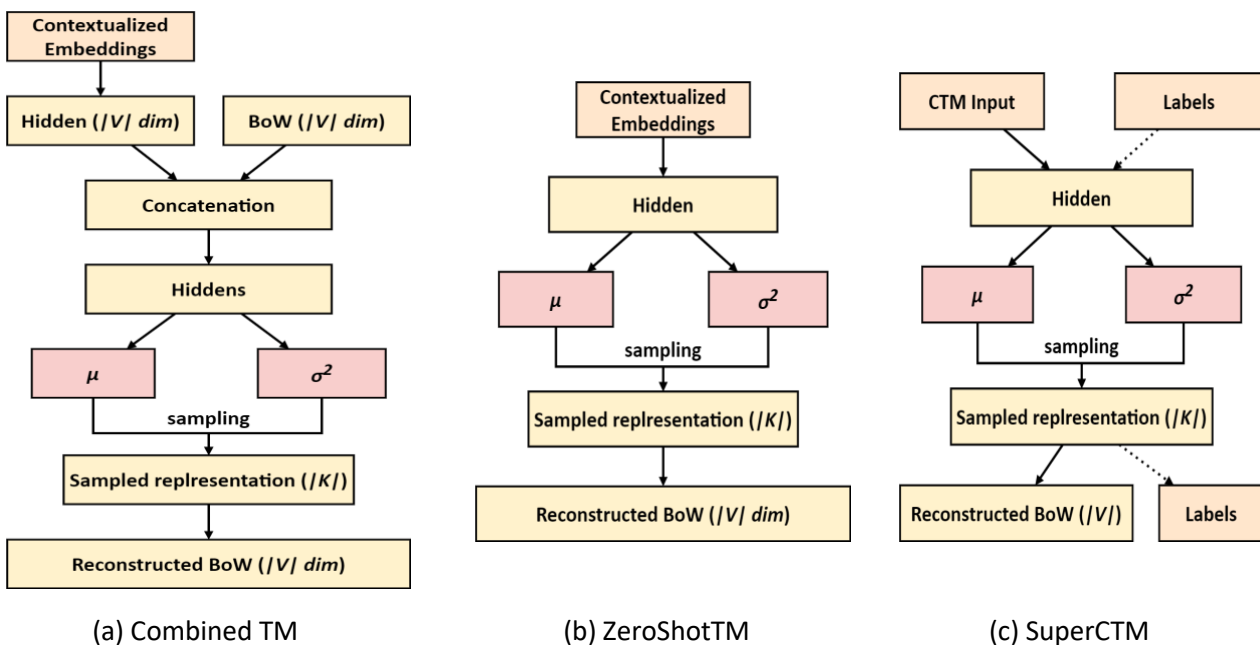
- *CombinedTM* combines the BoW representation already given as input to ProdLDA with SBERT embeddings, as shown in Figure 4(a). This process seems to increase the coherence of the predicted topics.
- *ZeroShotTM* utilizes only SBERT embeddings as input to the inference network (Figure 4(b)), thus making it adequate for zero-shot topic modeling and capable of dealing with unseen words at test time, since the BoW representation, which cannot account for missing terms, has been disregarded. Additionally, this NTM can be used for cross-lingual topic modeling, i.e., training and test sets’ languages do not need to be the same.
- *Supervised CTM (SuperCTM)* is inspired by the work from [20], in the sense that the authors implement their approach under the CTM architecture. Therefore, SuperCTM adds to either CombinedTM or ZeroShotTM a set of labels (Figure 4(c)) to give more information to the model, which should increase the correlation of the topics with respect to the labels.

- β -CTM implements the intuitions behind the work of [21] under the CTM architecture. The key idea is to apply a weight to the Kullback-Leibler loss function to help create representations by forcing independence in the components.

CTMs have one limitation inherited from the contextualized embeddings, namely that the contextualized representation of a document is limited to the pre-trained language model's sentence-length; that is, for the case of using SBERT embeddings, if the document length is larger than 512, the rest of the document will be discarded.

Note that CTMs perform better when the BOW size has been restricted to not exceed 2,000 elements. This is because the contextualized embeddings are projected into the vocabulary space, so the bigger the vocabulary, the more parameters are needed, thus hardening the training and making it prone to bad fitting.

Figure 4. High-level sketch of Contextualized Topic Models



Although the TMT includes in its implementation all the approaches that we have just reviewed, only the CombinedTM has been selected for its use in IntelComp, and this document will focus solely on the evaluation of this algorithm, since this is the architecture whose functionality can be more easily compared to that of the other topic modeling tools included. The preliminary tests that have been carried out suggest that CombinedTM provides topics with very high coherence, thus being a very interesting option as long as high training and processing times during inference are acceptable and/or the component is deployed with access to GPU hardware.

Two other approaches that could be of interest in certain situations are ZeroShotTM, which provides a multilingual model if an appropriate embedding model is used, and SuperCTM for those cases in which labels are available. The use of these tools, which, as mentioned, are

already available (though not directly accessible through the developed UIs) in the TMT, could be reconsidered based on the requirements posed by the users during the execution of the LLS.

Similar as for the AVITM implementations, the TMT offers the possibility of configuring a set of default and advanced settings; Table 5 summarizes the most important of these default parameters.

Table 5. Most relevant setting parameters for Contextualized Topic Modeling

Parameter	Default	Description
num_topics	25	Number of topics for the model. A large number of topics will provide a higher level of detail, and will allow us to find some very specialized topics of small size. One drawback, however, is that larger topics can appear repetitive and fragmented. On the contrary, a small number of topics is useful to have a first approximation of the main topics of the corpus. This value is commonly referred to as ‘number of components’ in the CTM implementation.
model_type	ProdLDA	The underlying AVITM implementation used for training, over which the contextualized topic model is constructed, namely ‘ProdLDA’ or ‘LDA’.
num_epochs	100	Number of complete passes through the training dataset over which the model will be trained. It can be set to an integer number between zero and infinity
batch_size	64	Number of samples in the mini batch for neural network training. It should have a value larger or equal to one and less or equal than the number of samples in the training dataset.

3.3.4. Construction of Hierarchical Topic Models

Previous analysis conducted by several IntelComp project partners has shown that the topics of a flat model (like the ones described in Subsections 3.3.1-3.3.3) do not possess the granularity expected by STI domain experts, thus making it necessary to perform a thematic analysis with different levels of resolution. Simply increasing the number of topics does not resolve this problem because in that case one typically ends up with redundant or unnecessary topics. Hence, we have provided within the TMT two novel approaches for assembling hierarchical topic models that allow the incorporation of domain experts’ knowledge into the hierarchy construction. The proposed schemes consist of:

1. Initial modeling of a selected dataset using a standard topic modeling algorithm. We will refer to this initial modeling as a level-1 topic model.
2. Nested modeling of user-selected first-level topics to increase the granularity of the analysis. We will refer to these submodels as level-2 topic models.

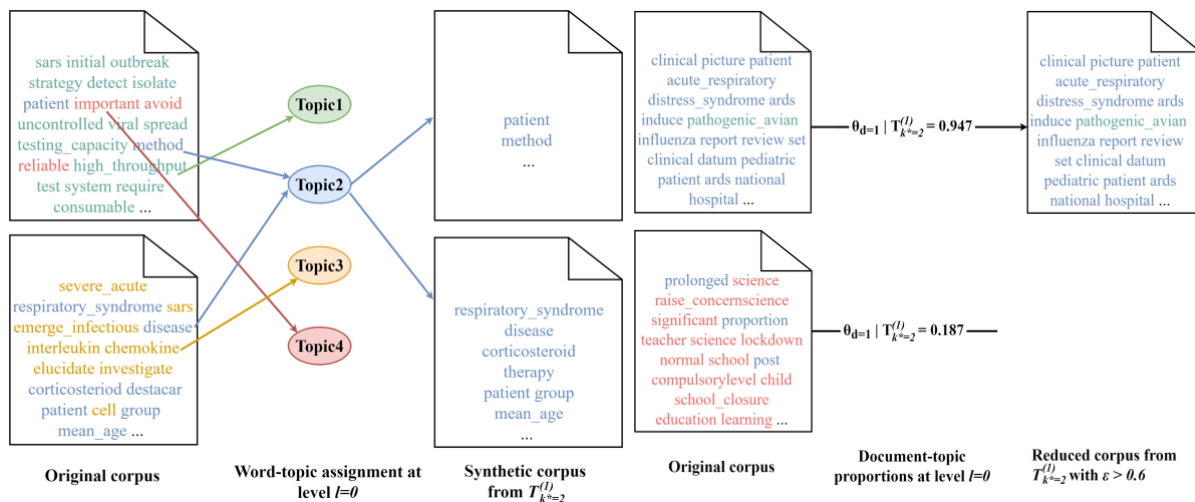
Although the process could be iterated, currently the TMT only supports hierarchical topic models with two levels. Both level-1 and level-2 topic models are trained using any existing topic modeling algorithm (for the current version of the TMT, one chosen between Mallet, SparkLDA, ProdLDA, or CTM). However, level-2 submodels are trained over specific training datasets that are dependent on the topic that is being expanded. Two different approaches for creating these

datasets can be used, giving place to the two novel solutions for hierarchical topic modeling: For HTM-WS (HTM with *word selection*) a synthetic corpus is generated by keeping the words from each document in the level-1 model that were assigned to topic being expanded, whereas for HTM-DS (HTM with *document selection*) a reduced training set is generated keeping only those documents of the original corpus whose weight for the selected topic is above a threshold. Table 6 summarizes the parameters implied in the construction of the HTMs, and Figure 5 provides a graphical representation of the corpus generation under each of the two HTM versions.

Table 6. Setting parameters for the construction of level-2 topics models

Parameter	Default	Description
htm_version	htm-ds	Hierarchical topic model schemes used for the generation of the level-2 model's training corpus, either 'htm-ws' or 'htm-ds'.
thr	0.2	Document-topic threshold that a document in the level-1 topic model's corpus must have to be kept in the level-2 model's corpus (only applies to HTM-DS).

Figure 5. Graphical representation of HTM-WS and HTM-DS corpus generation for fictional topic 2 (represented in blue)



3.4. Topic Modeling technologies incorporated in the TMT

One of the distinctive elements of the TMT is the inclusion of a series of tools designed to incorporate the prior knowledge and experience of domain experts into the topic models. In this way, it is intended both to obtain models of higher quality and more interpretable topics, as well as to align the trained models to the needs of the experts. To achieve these objectives, the toolbox incorporates two types of procedures:

1. *Topic model visualization tools*: they allow the user to explore the topic model and to know the nature of the obtained topics, their composition, the relationship between them, etc. This is fundamentally statistical information that may require a long time for its calculation, so the execution of these tools is carried out right after the training of a given model. In this way, although training times are lengthened, the user can navigate

more easily with the available models. Also note that increasing the training time does not have a significant impact on the usability of the tool, since this is a task that will take place asynchronously in any case.

2. *Curation tools*: they allow deletion or merge of topics, tagging them manually, deleting some of the words that characterize them, etc. This set of tools has been designed with the aim of being as real-time as possible, as their use is expected to take place in an interactive manner at the IMT. Although this set of tools can be very useful in order to obtain a more interpretable model, it must be considered that their use may degrade the internal coherence of the model, e.g., resulting in an inappropriate resizing of the topics of the model, and presence of "super-topics" with overestimation of their size in the corpus.

Use of curation tools is recommended only for the final adjustment steps of a model. Previously, it is recommended to train a series of initial auxiliary models, with the mere objective of identifying a series of stopwords and relevant equivalences for the corpus that is being analyzed. Schematically, to obtain high-quality models, it is recommended to iterate the following steps through the functionalities provided by the UIs:

- Obtaining a representative vocabulary: it is especially important to clean the terms that appear in the representation of the topics. For this, the following steps should be iterated:
 - Training of a topic model with a moderate number of topics (e.g., 30-40 topics). This training includes the preprocessing of the text prior to model training.
 - Vocabulary cleaning according to the following criteria:
 - uninformative terms for the corpus in question should be marked as stopwords.
 - terms that appear highlighted in several topics of different nature should be marked as stopwords.
 - Equivalent terms should be used to correct errors in the NLP pipelines, in particular it is important to correct lemmatization errors.
 - Equivalent terms should be used to map synonyms to a common form, including acronyms that have not been correctly identified by the Named Entity Recognition module.
- Curation of a final model with the desired number of topics. To obtain an easily interpretable model, it is recommended to slightly overestimate the number of topics in the model, and subsequently merge topics of a similar nature if necessary.

The remainder of the section briefly describes the visualization and curation tools available in the TMT to support these procedures. For the most part, these tools are available in the two available UIs: graphical and terminal-based.

3.4.1. Topic visualization and annotation

Visualization of topics statistics

A first tool for browsing the topics obtained consists of directly providing the list of topics that make up the model together with their description. As an example, Figure 6 includes the description of the first 13 topics of a model of 60 topics trained on the S2CS corpus (Papers of Semantic Scholar in the Computer Science category). The topics are ordered according to their size, and for each topic the following information is shown: 1) its relative size; 2) the corresponding label (automatically or manually generated); and 3) its description in terms of the most relevant words of the topic.

Figure 6. Topics description for a topic model of 60 topics of S2CS (Computer Science papers in Semantic Scholar). The first 13 topics are shown.

Topic ID	Size	Label	Word Description
0	0.0242	Artificial Intelligence generic	review, theory, understanding, science, intell...
1	0.0238	Probabilistic methods	estimation, method, distribution, probability,...
2	0.0232	Software development	software, process, product, project, engineeri...
3	0.0221	High parallel computing	parallel, memory, processor, architecture, har...
4	0.0217	Exploitation	management, business, organization, market, cu...
5	0.0216	Training	learning, education, teaching, skill, technolo...
6	0.0215	Logic semantics	logic, language, formal, semantic, specificati...
7	0.0207	Computer Vision	object, 3D, segmentation, shape, method, detec...
8	0.0206	Graph theory, Set Theory	graph, vertex, edge, polynomial, set, algorith...
9	0.0206	Platform services	service, architecture, management, web, platfo...
10	0.0199	Generic publication terms	library, university, science, journal, interna...
11	0.0191	Optimization	algorithm, cluster, optimization, genetic, sea...
12	0.019	Machine learning	classification, detection, feature, machine, l...

The tool also calculates other additional statistics for each topic, which are only shown to facilitate the development of certain operations. Its general inclusion could be considered, although in this first version of the TMT it has been considered more sensible to omit such statistics so as not to overwhelm the user with possibly unnecessary information:

- The description (word distribution) of the topic with a penalty for the most common terms. When obtaining the description of a topic, it is usual to select the words with the highest frequency of appearance. An alternative that is actually used by default in the TMT is to penalize the representation of those terms that appear with high probability in a large number of topics (similarly to the penalty according to *Inverse Document Frequency* applied to the BOW representation of a text).
- Number of documents in which each topic is present (*N Active docs*): this value gives an idea of the level of use of a certain topic throughout the entire corpus, so that those topics that are more widespread and have a more horizontal character, are therefore less semantically discriminating. Noisy topics are expected to trigger a high number of topics, so this information will be provided when the users request to delete topics from the model.
- Entropy: Gives an idea of whether a topic is characterized by a reduced number of terms or by a broad set (each of them in a smaller proportion). More information about this statistic is shown in Section 3.5. This statistic was originally calculated also with the aim

of helping in the detection of generic or noisy topics, but we found that the number of active docs was a more useful indicator for that.

- Coherence: Gives an idea of the degree of cohesion of the high probability terms for a given topic. More information about this statistic is shown in Section 3.5. The coherence of topics can be used as an indicator to help the user decide which topics are good candidates to be further split.

Automatic labeling of topics

Once the topics are generated, labels that may describe those topics are automatically generated to ease the interpretation of the topics. If necessary, the user will be able to later modify and curate the labels provided by the automatic system.

The automatic labeling system assigns labels taken from a list of feasible labels that can be created by the user (similarly to lists of stopwords or keywords). The current implementation of the system incorporates a set of labels taken from the collection of categories available in wikipedia.

For the Automatic Topic Labeling itself, i.e., the system that assigns to each collection of words characterizing one of the topics in the model a specific label from the list, a zero-shot-classifier available at HuggingFace is used. Zero-shot-classifiers are an excellent technology for those cases in which the set of categories is known but no labeled data is available for supervised training. More information on this kind of classifiers can be found in IntelComp deliverable D3.4. - Classification Service.

To be more specific about the solution implemented, we have used a HuggingFace pipeline defined by two essential arguments, the classifier type ("*zero-shot-classification*") and the model itself ("*facebook/bart-large-mnli*")⁶. This model has proved to perform well in most cases both in terms of time and accuracy.

For example, given the following sentence: "one day I will see the world", and candidate labels ['travel', 'cooking', 'dancing'], the output would contain a score for each label {'labels': ['travel', 'dancing', 'cooking'], 'scores': [0.994, 0.003, 0.003]...}, from which the final label assigned to the topic would be "travel".

Stopwords suggestion tool

As has already been mentioned, the TMT aims to make it easier for the user both to obtain a final high-quality topic model which can be taken to production, and to obtain auxiliary models that help in the task of identifying potential stopwords that should be eliminated. To help with the latter task, the software incorporates a functionality that suggests to the user a list of candidate terms to be considered as stopwords based on the following rules:

⁶ The full description, use cases and interactive examples can be found in its original HuggingFace webpage: <https://huggingface.co/facebook/bart-large-mnli>.

- The user initially selects a series of topics that are considered to be of little semantic value (e.g., noisy topics, too generic topics, irrelevant topics, etc)
- The user also provides a probability threshold to select the terms to be included in the tool's analysis.
- The tool explores the appearance of the terms present in the topics selected by the user in the rest of the topics of the model. Based on this, it provides two lists:
 - a first list with the terms that only appear in noisy or irrelevant topics, and that can be considered as stopwords with great probability (the user can then choose to directly create a list of stopwords with these terms, or to carry out a manual inspection).
 - a second list of terms that are not suggested as stopwords, as those are terms that appear in other topics of the model. The user is recommended to monitor at least the first terms of the list, in search of possible additional stopwords.

pyLDAvis

A graphical visualization of the topic model is provided through the LDAvis tool developed by Carson Sievert and Kenny Shirley⁷, making use of the pyLDAvis Python library. In order to obtain a representation of sufficient quality, but that can be calculated quickly, the visualization is calculated on a maximum of 10,000 documents taken at random from the corpus, so the visualization is not deterministic and could change slightly with each run for a fixed model.

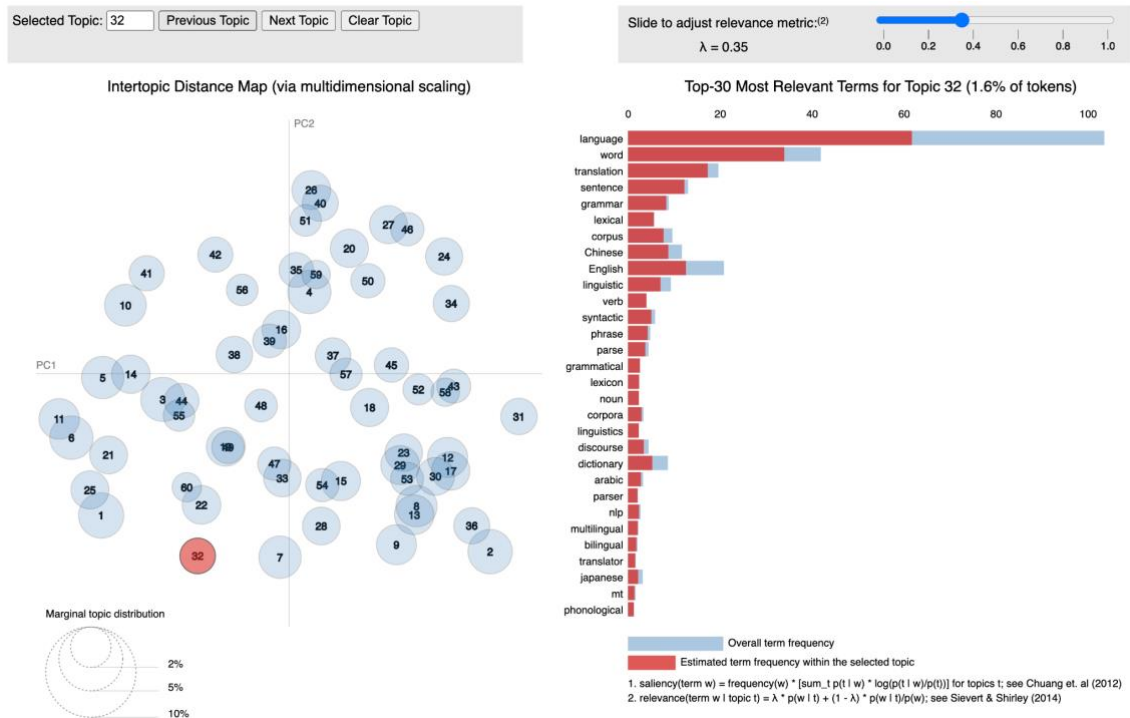
An example of such visualization is provided in Figure 7. In the subfigure on the left there is a map of topics, in which each topic is represented with a bubble whose size is proportional to that of the topic. Similar topics appear close together in the figure. A more or less homogeneous distribution of the topics covering the entire two-dimensional mapping space, suggests that the optimization of the model has been carried out correctly.

For the analysis of each topic, the user can select directly from the navigation tool at the top, or by clicking on the desired topic, and the right subfigure will show a histogram of the terms, with their frequency of appearance in the selected topic and in the complete corpus. The sliding bar that is provided in the upper right part (parameter λ) allows us to measure the relevance of each term for a given topic, penalizing those terms that appear in a large number of topics.

The pyLDAvis tool requires an external browser, so users of the terminal-based UI must directly access the generated file available in the model folder. In the case of the graphical interface, the HTML file is embedded directly, facilitating user navigation.

⁷ <https://github.com/cpsievert/LDAvis>.

Figure 7. pyLDavis output for a topic model of 60 topics of S2CS (Computer Science papers in Semantic Scholar). Topic 32 (NLP) has been selected.



3.4.2. Topic curation tools

We present in this subsection a set of tools designed to improve the quality of the final model.

Manual labeling of topics

Although the toolbox includes an automatic topic annotation tool, the functionality of such tool is far from perfect, and the user may wish to annotate one or several topics of the model according to their preferences. To this end, the user is provided with the possibility of providing a text-free label, or resorting to the description of the topic based on the concatenation of the most relevant terms for the topic in question. This decision can be made on a topic-by-topic basis, that is, the user could keep the automatic labels for a subset of the topics, and provide a manual label or keep the word list individually for each of the other topics in the model.

Suggestion of similar topics

As a previous step to merging the model's topics, the toolbox provides the option of exploring pairs of topics that can be merged because they are considered semantically close. To do this, two different criteria have been implemented:

- First, the co-occurrence of topics in the corpus documents is analyzed, i.e., topics that activate the same documents are searched for. For this, each topic is represented as a vector with length equal to the number of documents, with the weights of each document for the topic in question, and the correlations between the vectors thus

constructed are analyzed. In this way, the topics with the highest correlations are proposed as possible candidates to be merged.

- Secondly, we directly exploit the composition of the topics in terms of the words that characterize them. Therefore, we measure the distance between the rows of matrix β , and we select as candidates those topics that have a small distance between them. Since we are analyzing distances between vectors that represent probability distributions, the tool applies the Jensen-Shannon distance as the selected criterion.

Figure 8 below shows the most similar pairs of topics for a particular topic model. The user of the TMT can then use or ignore these suggestions to decide which topics should be merged. For taking a more informed decision, the user can also combine suggestions provided using both criteria with location of topics in the pyLDavis tool.

Figure 8. Suggestion of most similar pairs of topics for a topic model of 60 topics of S2CS (Computer Science papers in Semantic Scholar).

```

*****
Most similar topic according to topic cooccurrence in the documents
(Be aware that Topic IDs will change after every deletion or fusion)
*****

=====
Correlation between topics 23 and 57: 13.35%
Label                                     Word Description
-----
Topic ID
23                                     Communication systems  channel, transmission, communication, receiver...
57      code, error, decoder, relay, rate, channel, en...  code, error, decoder, relay, rate, channel, en...
=====
Correlation between topics 16 and 33: 11.21%
Label                                     Word Description
-----
Topic ID
16                                     Non-linear systems    non-linear, stability, linear, dynamics, state...
33      control, controller, motor, adaptive, simulati...  control, controller, motor, adaptive, simulati...
=====
Correlation between topics 33 and 45: 9.84%
Label                                     Word Description
-----
Topic ID
33      control, controller, motor, adaptive, simulati...  control, controller, motor, adaptive, simulati...
45      power, voltage, converter, wind, load, switch,...  power, voltage, converter, wind, load, switch,...
=====

*****
Most similar topic according to topic word-based description
(Be aware that Topic IDs will change after every deletion or fusion)
*****

=====
Correlation between topics 19 and 25: 47.37%
Label                                     Word Description
-----
Topic ID
19                                     Networks              network, node, wireless, sensor, rout, protoco...
25      network, packet, traffic, protocol, service, w...  network, packet, traffic, protocol, service, w...
=====
Correlation between topics 5 and 59: 45.19%
Label                                     Word Description
-----
Topic ID
5                                     Training             learning, education, teaching, skill, technolo...
59      teaching, read, write, learning, English, stud...  teaching, read, write, learning, English, stud...
=====
Correlation between topics 1 and 16: 44.69%
Label                                     Word Description
-----
Topic ID
1       Probabilistic methods  estimation, method, distribution, probability,...
16      Non-linear systems    non-linear, stability, linear, dynamics, state...
=====
    
```

Topic model edition

Finally, we include a series of procedures for modifying the very matrices that define the topic model:

- Fusion of topics. The user selects the topics that must be merged, and the tool carries out a series of actions to guarantee the probabilistic integrity of the model. In the first place, the vector representation of the new topic, obtained as a fusion of the selected ones, is obtained as a weighted average of the representation of the existing topics. Second, the weight that corresponds to each document for the new merged topic is obtained as the sum of the weights of the original topics. Finally, statistics such as entropy, number of documents in which the merged topic is active, description, etc, must be calculated from scratch.
- Sorting out the topics of the model: it consists of a simple operation of reordering the topics according to size, that allows the largest topics to be placed first. This operation is of special interest after the merge of topics.
- Topic deletion. The user provides a list of topics to remove, and the tool removes the corresponding rows from matrix β , and the corresponding columns from matrix θ , followed by normalization of the rows in θ . It should be mentioned that these operations can be useful to eliminate topics of little interest from the model, but they result in the loss of the internal coherence of the model, since the calculation of the representation of each document is carried out in a simplistic way, i.e., proportionally resizing the weights of all the topics, without taking into account that not all the topics will be equally related to the eliminated topic, and therefore the proportional reallocation of weights could not be correct.
- Topic model reset. If the users find that the accumulation of included changes has led to an unwanted model, they can choose to discard all changes, and return to the original model obtained after the initial training.

3.5. Evaluation of topic models

In this subsection we will describe some of the metrics that have been incorporated to the TMT, under which the different topic model technologies can be compared. We focus just on metrics for topic model assessment with respect to the quality of the extracted topics, not the scalability of the implementations.

- *Log-likelihood*. The log-likelihood score of a set of unseen documents measures how probable such data given a learned topic model is, that is, the model's ability to replicate the statistics of the held-out documents. Therefore, the larger the log-likelihood score associated with a topic model is, the better the model is considered.

One problem associated with this score is that for some models, such as LDA, it may be intractable to compute, thus making it necessary to approximate the probability, as shown by [22].

- *Perplexity*. It measures the ability of a trained topic model to predict previously unseen data (i.e., held-out documents). As the perplexity consists of a decreasing function of the likelihood of new documents, there is an inverse proportional relationship between the likelihood of the words appearing in new documents and the perplexity. Hence, a

“good” topic model should be characterized by a low perplexity score, as it would imply that it can form reliable predictions for words appearing in a new document.

Note [23] showed that perplexity and the human judgment of topics generated by topic models do not correlate, meaning that, as the perplexity score improves, the human interpretability of the topics’ coherence can actually get worse.

Perplexity’s weaknesses include the inability of capturing contextual information, i.e., the relationship between words in a topic or topics in a document.

- *Coherence*. It measures how semantically cohesive the inferred topics of a topic model are. The conditioned probability between two terms (i.e., probability of a word to occur given the other) is used to calculate the coherence score, thereby giving a certain context to all the words conforming to a topic, as all topic’s terms will be related to a certain degree.

There exist multiple coherence measures which calculate the score under different assumptions. Among them, it is worth mentioning the ones included in Table 7, but for the sake of the evaluation of the TMT we will only utilize *Umass* and C_{NPMI} .

Table 7. Most relevant coherence metrics in the literature

Coherence	Paper presented
C_{UCI}	[24]
U_{mass}	[25]
C_{NPMI}	[26]
C_v	[27]
C_{WE} (WETC)	[28], [12]

While *Umass* is based on the co-occurrence of the words within the documents being modeled and does not rely upon any external reference corpus, C_{NPMI} utilizes the normalized pointwise mutual information (NPMI) for the calculation of the coherence score and does rely on an external source. By using these metrics, we can evaluate both the capability of the topic models for correctly identifying words that frequently co-occur in documents as a topic (i.e., the topics represent the corpus accurately) and the frequency of the words appearing together in alignment with a purported assessment of consistency in the hands of an expert.

- *Entropy*. By definition, it is a measure of the average amount of uncertainty or information that is intrinsic to each random variable assignment. In particular, for this project, we will consider document entropy and topic entropy.

If the topics belonging to a topic modeling outcome are evenly distributed across many documents, the model is considered to have a high document entropy; otherwise, the model has a low entropy when the topics are just concentrated in a few documents.

Similarly, if a topic is composed of most of the words present in the vocabulary, we are dealing with a high topic entropy, while a topic that comprises just a few words will have a low topic entropy.

- *Inverted Rank-Biased Overlap (Inverted RBO)*. Metric that evaluates the diversity of the topics generated by a single model. Exploited in [12], it is defined as the reciprocal of the standard RBO [29], [30]. RBO is based on a probabilistic model in which the overlaps between the top word of two topics is calculated, by allowing the topics to be composed of different words and utilizing weighted ranking (i.e., lists that share the same words but with different rankings are less penalized than two lists sharing the same words at the highest rank. This measure ranges between 0 and 1, where 0 is indicative of the topics under study being identical.

4. SOFTWARE DOCUMENTATION

In this section we document the TMT software itself. We will start by describing the architecture of the repository and the documentation associated with it. The dependencies of the software and the configuration file whose settings must be adjusted depending on the characteristics of the environment in which the topic modeling is going to be executed are also described.

The execution of the software is based on “projects”, so that different projects can coexist with their own data, models, etc. Each project is associated with a directory whose organization is briefly described. We conclude the documentation about the software by describing the access to the functionality implemented by the different components that have been dockerized, as well as the Python front-ends developed on the services.

4.1. Software architecture

The software package can be downloaded from the IntelComp repository in GitHub⁸. It contains all software modules and its documentation. The documentation has been generated using Sphinx in ReadTheDocs format and is available under the /docs folder of the project. Figure 9 shows a snapshot of the page describing one of the components of the project.

The overall organization of the software is described below. Next to each folder we provide a brief description of its purpose, indicating relevant files or subfolders when necessary:

⁸ <https://github.com/IntelCompH2020/topicmodeler>.

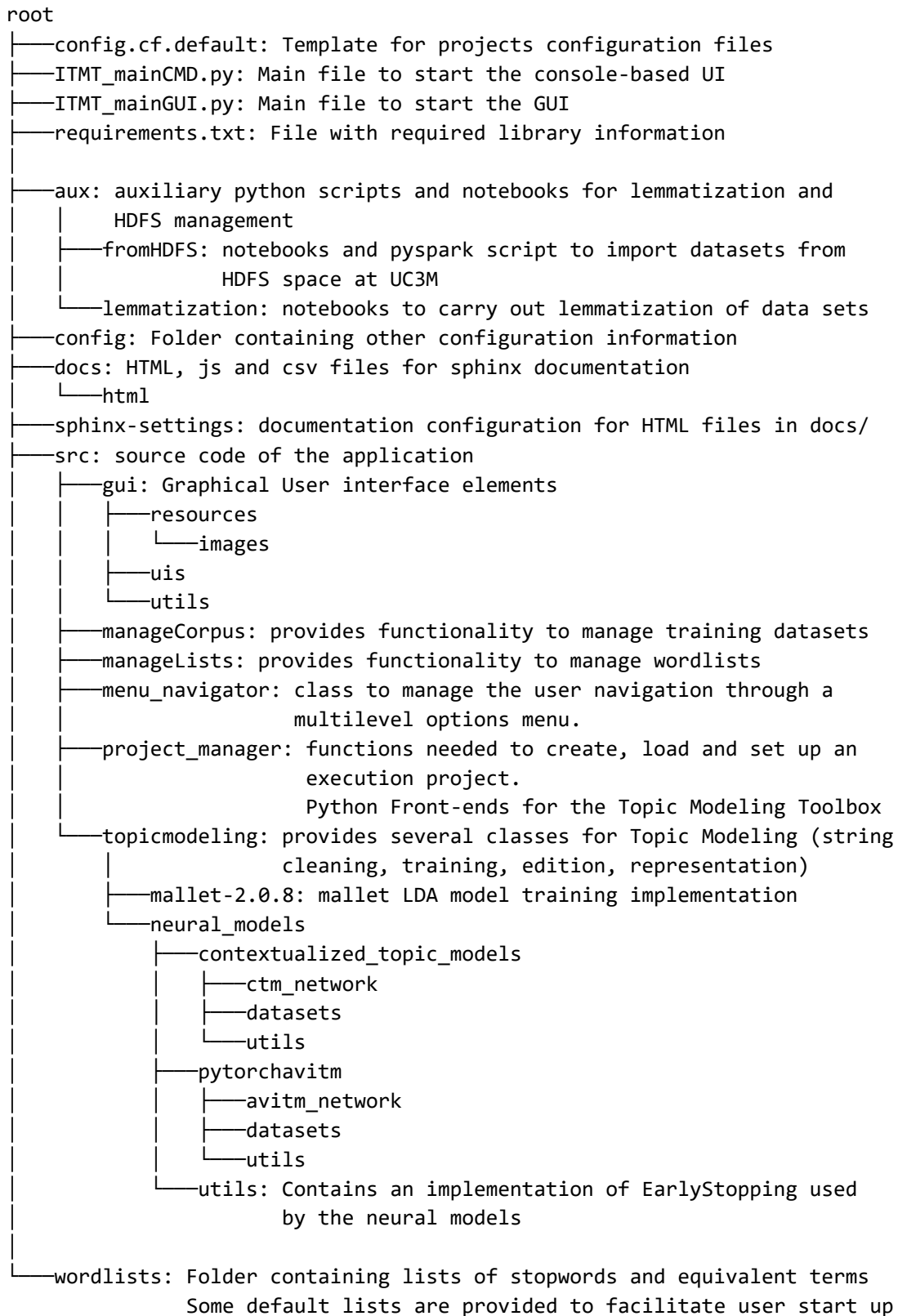


Figure 9. Snapshot of the software documentation

src.manageLists.manageLists

- IntelComp H2020 project

Contains the class implementing the functionality required by the Interactive Model Trainer for creating lists of

- keywords
- equivalent terms
- stopwords

class src.manageLists.manageLists.ListManager [\[source\]](#)

Bases: `object`

Main class to manage functionality for the creation, edition, etc of lists of stopwords/keywords/equivalent_terms

listWordLists(path_wordlists) [\[source\]](#)

Returns a dictionary with all wordlists available in the folder

Parameters: `path_wordlists` (*pathlib.Path*) – Path to the folder hosting the wordlists

Returns: `allWdLists` – One dictionary entry per wordlist key is the absolute path to the wordlist value is a dictionary with metadata

Return type: Dictionary (path -> dictionary)

createList(path_wordlists, WdList) [\[source\]](#)

Saves a (logical) training dataset in the indicated dataset folder

Parameters:

- `path_wordlists` (*pathlib.Path*) – Path to the folder hosting the wordlists
- `WdList` – Dictionary with WordList

Returns: `status` –

- 0 if the wordlist could not be created
- 1 if the wordlist was created successfully
- 2 if the wordlist replaced an existing one

Return type: int

The main components of the folder structure in the software repository are:

- `docs/`: Documentation folder.
- `src/`: The python software package. Contains all classes and methods.
- **Main scripts**: two executable python scripts that can be used to test all software modules through a terminal/command window (`ITMT_mainCMD.py`) or through a PyQt6 GUI (`ITMT_mainGUI.py`).

The classes and methods in folder `src/` are structured in several modules:

- Modules related to the topic modeling back-end services. These are the key components that implement the back-end services provided by the TMT, responsible for the creation of training corpus and training of topic models. These services have been dockerized and are the only ones that are needed by the IMT operation. The corresponding command line API for using the services provided by these components is described in Subsection 4.5. Here, we describe the general purpose of each component:
 - `manageCorpus/`: Contains a series of utilities to 1) download data sets available as parquet files from an HDFS, 2) create training data sets from the downloaded datasets, and 3) standard operations for listing, removing, renaming, etc, the generated training datasets.

- `manageLists/`: Contains a series of utilities to work with lists of words representing stopwords, equivalent terms, keywords, or labels for the automatic topic labeling service. Provided methods allow creation, edition and removal of wordlists.
- `topicmodeling/`: This folder contains two different components:
 - `topicmodeling.py`: implements 1) text preprocessing for generating the data sets in the format needed to train the topic models, and 2) the training of the topic models.
 - `manageModels.py`: implements 1) general operations on topic models (e.g., listing of models, removing or renaming them), and 2) the service to represent, visualize, and curate topic models, independently of the algorithm that was used to train them.
- Modules related to the Python front-ends:
 - `gui/`: contains all classes and methods related to the GUI.
 - `menu_navigator/`: Contains the class `MenuNavigator`, which reads and interprets the menu structure (defined in a configuration file) that will be used by the console-based UI.
 - `project_manager/`: Contains the classes implementing the operations to create a new project and, most importantly, the logic of operations for both the console-based and graphical UIs. Some important classes for the operation of the UIs are:
 - class `ITMTTaskManager`: Centralizes the use of the back-end services. The methods in this class compose the commands that need to be run to parse the input and output that is required by the services.
 - class `ITMTTaskManagerCMD`: Derived from the previous, it manages user interaction in the console-based UI. I.e., for each method the class will retrieve the user required information, then call the corresponding method in the parent class to execute the operations required by the user, and finally parse the results that need to be provided back to the user.
 - class `ITMTTaskManagerGUI`: Also derived from class `ITMTTaskManager`, it provides extra functionality to the task manager, to be used by the GUI.
 - `utils/`: Contains some useful auxiliary methods used by the console-based UI, e.g., to request user information, confirmation of actions, etc.

4.2. Requirements

The software has been developed mostly in Python, and so, several libraries are required to execute the code. They can be found in the *requirements.txt* file, and installed using the following command in a terminal:

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

Table 8. List of libraries used by the Topic Modeling Toolbox. For each required library we indicate the version that has been used for the current release and their licenses

Library	Version	License
PyQt6	6.3.0	GPL v3
PyQt6-Qt6	6.3.0	LGPL v3
PyQt6-WebEngine	6.3.0	GPL v3
PyQt6-WebEngine-Qt6	6.3.0	LGPL v3
PyQt6-sip	13.3.1	SIP
Sphinx	5.0.2	BSD
colored	1.4.2	MIT
dask	2022.7.0	BSD
fastparquet	0.8.1	Apache Software
gensim	4.0.1	LGPL-2.1-only
ipython	8.0.1	BSD
ipywidgets	7.6.5	BSD
langdetect	1.0.9	Apache Software
matplotlib	3.5.1	Python Software Foundation
nltk	3.7	Apache Software
numpy	1.22.4	BSD
pandas	1.4.1	BSD
pyLDavis	3.3.1	MIT
pyarrow	8.0.0	Apache Software
pyspark	3.3.0	Apache Software
regex	2021.11.2	Apache Software
scikit-learn	1.0.2	new BSD
scikit-optimize	0.9.0	BSD
seaborn	0.11.2	BSD
sentence-transformers	2.2.2	Apache Software
sphinx-rtd-theme	1.0.0	MIT

torch	1.10.2	BSD
tqdm	4.62.3	MIT; MMPL 2.0

For the development of the TMT and, in particular, for the inclusion of the neural topic modeling tools, we have resorted also to third-party implementations, making use of a number of Github repositories as listed below (both licensed under MIT license):

- <https://github.com/estebandito22/PyTorchAVITM>
- <https://github.com/MilaNLPProc/contextualized-topic-models>

4.3. Configuration file

The root folder of the project must contain a file with the settings for the application to work correctly in each specific environment. Such configurations include paths to directories/files, model parameters, etc. Default settings are provided in file *config.cf.default* available in the Github repository. After creating a project folder, such file should be copied in the root folder of the project under the name *config.cf*, and configuration settings modified as needed.

The configuration parameters are described next. Default values included in file *config.cf.default* are generic values that are known to work well in general situations, and are widely used in the scientific literature or open-source software libraries.

Section logformat: specify format for the log outputs		
Parameter	Description	Default value
filename	Destination file for the logging messages	msgs.log
datefmt	Timestamp format	%%Y-%%d-%%m %%H:%%M:%%S
file_format	Format for the log messages written to file	%%(asctime)s %%((levelname)-8s %%((message)s
file_level	Minimum level for the log messages written to file	INFO
cons_level	Minimum level for the log messages shown in the console	DEBUG
cons_format	Format for the log messages displayed in the console	%%((levelname)-8s %%((message)s
Section Spark:		
Parameter	Description	Default value
spark_available	Whether spark is available	True
machines	Number of machines to include in Spark cluster (this is dependent on how	10

	Spark tasks are submitted in UC3M infrastructure)	
cores	Number of cores per Spark node (this is dependent on how Spark tasks are submitted in UC3M infrastructure)	4
script_spark	Location of script to execute spark notebook	/export/usuarios_ml4ds/jarenas/script-spark/script-spark
token_spark	Token to execute spark	/export/usuarios_ml4ds/jarenas/script-spark/tokencluster.json
Section Dask:		
Parameter	Description	Default value
num_workers	Number of workers to use in the Text Preprocessing for Topic Modeling when recurring to Dask. Should be less than the available number of cores. Use 0 to use Dask default value (i.e., number of available cores).	0
Section HDFS: These paths should be adjusted to each particular deployment		
Parameter	Description	Default value at UC3M
Semantic Scholar	Parquet table for the Semantic Scholar dataset	/export/ml4ds/IntelComp/Datalake/SemanticScholar/20220201/papers.parquet
PATSTAT	Parquet table for PATSTAT	/export/ml4ds/IntelComp/Datalake/PATSTAT/2022_Spring/patstat_appln.parquet
CORDIS	Parquet table for the CORDIS dataset	/export/ml4ds/IntelComp/Datalake/CORDIS/20220221/new_parquet/projects.parquet
Section Preproc: Text Preprocessing for preparing the datasets for Topic Modeling		
Parameter	Description	Default value
min_lemas	Minimum number of words to keep document in corpus	15
no_below	Remove words with less than no_below occurrences	10
no_above	Remove words appearing in more than a given percentage of documents	0.6
keep_n	Maximum number of words in vocabulary	500.000
Section TM: Common settings parameters for all Topic Modeling algorithms		

Parameter	Description	Default value
ntopics	Default setting for number of topics	25
thetas_thr	Threshold for topic activation in a doc (sparsification)	3e-3
Section Mallet™: Parameters for Mallet Topic Modeling		
Parameter	Description	Default value
mallet_path	Path to mallet binary	./mallet-2.0.8/bin/mallet
token_regexp	Regular expression for token identification	<code>[\\p{L}\\p{N}][\\p{L}\\p{N}\\p{P}]*\\p{L}</code>
alpha	Sum over topics of smoothing over doc-topic distributions	5
optimize_interval	Number of iterations between reestimating dirichlet hyperparameters	10
num_threads	Number of threads for parallel training	4
num_iterations	Number of iterations of Gibbs sampling	1000
doc_topic_thr	Threshold for topic activation in a doc (mallet training)	0
num_iterations_inf	Number of iterations	100
Section SparkLDA: Parameters for SparkLDA		
Parameter	Description	Default value
alpha	Hyperparameter for the Dirichlet distribution responsible for generating the topic vectors of the documents.	5
maxIterations	Maximum number of iterations for the optimization algorithm	20
optimizer	The algorithm used for algorithm optimization, either 'em' or 'online'	online
optimizeDocConcentration	Whether the doc-concentration parameter alpha will be optimized during the training of the algorithm	True
Section ProdLDA: Parameters for ProdLDA		
Parameter	Description	Default value
n_components	Number of topic components	10

model_type	Type of the model that is going to be trained, 'prodLDA' or 'LDA'	prodLDA
hidden_sizes	Size of the hidden layer	(100,100)
activation	Activation function to be used, chosen from 'softplus', 'relu', 'sigmoid', 'leakyrelu', 'rrelu', 'elu', 'selu' or 'tanh'	softplus
dropout	Percent of neurons to drop out	0.2
learn_priors	If true, priors are made learnable parameters	True
lr	Learning rate to be used for training	2e-3
momentum	Momentum to be used for training	0.99
solver	NN optimizer to be used, chosen from 'adagrad', 'adam', 'sgd', 'adadelata' or 'rmsprop'	adam
num_epochs	Number of epochs to train for	100
reduce_on_plateau	If true, reduce learning rate by 10x on plateau of 10 epochs	False
batch_size	Size of the batch to use for training	64
topic_prior_mean	Mean parameter of the prior	0.0
topic_prior_variance	Variance parameter of the prior	None
num_samples	Number of times the theta needs to be sampled	10
num_data_loader_workers	Number of subprocesses to use for data loading	0

Section CTM: Parameters for Contextualized Topic Models

Parameter	Description	Default value
num_topics	Default setting for number of topics	10
model_type	Type of the model that is going to be trained, 'prodLDA' or 'LDA'	prodLDA
ctm_model_type	'CombinedTM', 'ZeroShotTM'	CombinedTM
hidden_sizes	Size of the hidden layer	(100,100)
activation	Activation function to be used, chosen from 'softplus', 'relu', 'sigmoid', 'leakyrelu', 'rrelu', 'elu', 'selu' or 'tanh'	softplus
dropout	Percent of neurons to drop out	0.2

learn_priors	If true, priors are made learnable parameters	True
batch_size	Size of the batch to use for training	64
lr	Learning rate to be used for training	2e-3
momentum	Momentum to be used for training	0.99
solver	NN optimizer to be used, chosen from 'adagrad', 'adam', 'sgd', 'adadelata' or 'rmsprop'	adam
num_epochs	Number of epochs to train for	100
num_samples	Number of times the theta needs to be sampled	10
reduce_on_plateau	If true, reduce learning rate by 10x on plateau of 10 epochs	False
topic_prior_mean	Mean parameter of the prior	0.0
topic_prior_variance	Variance parameter of the prior	None
num_data_loader_workers	Number of subprocesses to use for data loading	0
label_size	Number of total labels	0
loss_weights	Dictionary with the name of the weight parameter (key) and the weight (value) for each loss	None
sbert_model_to_load	Model to be used for calculating the embeddings. Available models can be checked here.	paraphrase-distilroberta-base-v1

Hierarchical: Parameters for the hierarchical topic models

Parameter	Description	Default value
expansion_tpc	Topic in the 1-level model that is going to be expanded for the generation of the 2-level model's training corpus.	0
htm_version	Hierarchical topic model algorithm under which the 2-level model's training corpus is going to be generated, 'htm-ws' or 'htm-ds'.	htm-ds
thr	The proportion of the expansion topic documents that a document must have to be kept in the 2-level model's training corpus. Only used when the htm_version is 'htm-ds'.	0.2

4.4. Topic Modeling project description

4.4.1. Input folders

So as the system to work, **three input folders** must be provided, namely the **project folder** which serves the purpose of giving persistence to the TMT's outputs, and a set of **external folders** that provide the TMT with the required inputs, in addition to storing reasons, as it will be below explained.

Project folder structure

The project folder (hereafter referred to as **project_folder**) consists of the folder in which all the results related to the application's execution are saved. It must be provided at starting time by specifying an existing location, or a location in which it is desired to locate it, and is indispensable for the application's functioning at the current development state. In case it does not exist, the application creates it at the specified location, along with the file and folder structure required to store the output files. This folder hosts the following items:

- **datasets/**: It will store a json file for each (logical) training dataset created by the user during the application execution. Each json file will contain all the information that is necessary for the creation of the corpus at the training phase.
- **TMmodels/**: It contains all the topic models created by the user. A subfolder will be created for each trained topic model. Within these (sub)folders, we can distinguish several common elements between all topic modeling algorithms, namely:
 - **trainconfig.json**: A json file containing all the information necessary for the training of the model, as described in Subsection 4.4.2.
 - **vocabulary.txt**: Text document with the training corpus vocabulary.
 - **TMmodel/**: It stores all the required files for the curation of the topic model.

In case the vocabulary is created with the `CountVectorizer Spark` object, only the **vocabulary.txt** file described above will be created; otherwise, when the parallelization is carried via Dask and thus the vocabulary construction is done using Gensim dictionaries, an additional file **dictionary.gensim** will be created; this file contains the Gensim dictionary of the corpus with which the model was trained.

For those cases where deployment to a Spark cluster is not available, we have implemented a version of the pipeline based on Dask dataframes⁹, which is a transformation-based preprocessing pipeline enabling parallelization of the operations, thus, catering for better use of computational resources and more efficient memory management. This implementation allows us to take advantage of the available resources when the container is deployed on a virtual machine. However, the vocabulary construction is done using Gensim dictionaries, which requires sequential

⁹ <https://www.dask.org/>. The concept of a Dataframe in Dask is similar to Pandas, but Dask allows dataframe partitioning.

processing of the documents, thus, this procedure being considerably slower than the corresponding implementation in Spark.

Additionally, the following folders are noteworthy:

- `modelFiles/`: It stores all the files generated during the training of the model itself. Except for the *'thetasDist.pdf'* (file containing a figure in which the effect of thresholding in the thetas distribution is illustrated), the content of this directory varies depending on the algorithm used for the training of the model. In case Mallet was used, it stores, in addition, all the output files generated by Mallet; if a neural model is used, this folder only contains one extra file, namely, the corpus used for training of the model saved in text format¹⁰.
- either a file named `corpus.txt` or a folder named `corpus.parquet`, depending on which topic modeling algorithm is used for training (`corpus.txt` for Mallet, `corpus.parquet` for the neural models and SparkLDA).

Note that in case a level-2 model is generated for any of the level-1 models, a new folder containing the same information as described above will be created within the level-1 model's folder. For example, if we had a theoretically level-1 model (*Mallet-25*) trained with Mallet, from which a level-2 model (*Sub-Mallet-25-WS-topic4-10*) has been generated, also using Mallet, its folder structure would be the following:

```
Mallet-25
├── trainconfig.json
├── dictionary.gensim
├── vocabulary.txt
├── TMmodel/
├── modelFiles/
├── corpus.txt
└── Sub-Mallet-25-WS-topic4-10
    ├── trainconfig.json:
    ├── dictionary.gensim
    ├── vocabulary.txt
    ├── TMmodel/
    ├── modelFiles/
    └── corpus.txt
```

Apart from the previous folders and files, related to training datasets and topic models, a series of other files need to be included in the project folder for configuration purposes:

- `config.cf`: the local configuration file of the project, that can be modified by the user manually or through the GUI.

¹⁰ Note here that for the neural models we make a distinction between training and validation corpus. Here we are only saving in a text file the corpus used for training, as it is the one utilized for the extraction of the actual topic modeling results and the generation of the 2-level submodels' corpus.

- `config.cf.default`: the configuration file of the project with the default values. It cannot be modified by the user, so it is only used for being able to restore the changes made in the `config.cf` to their initial state.
- `metadata.yaml`: a file with metadata that stores the status of the project.
- `msgs.log`: log file of the latest code execution with this project.

External folder's structure

As external folders, i.e., folders that need to be provided as input to the application for it to work, we consider two, one containing the local data sets that are assumed to have been downloaded from the data catalog (`parquet_folder`), and a second one for storing the set of wordlists customized by the user (`wordlists_folder`).

The `parquet_folder` must be characterized by the following structure:

- `<dataset>.parquet`, where `<dataset>` is the name of a local dataset. The `parquet_folder` must contain as many `<dataset>.parquet` folders as datasets have been downloaded, and each of the folders will be composed of a set of parquet partitions with the data.
- `datasetMeta.json`: Json file containing the information about each of the downloaded datasets, so it must contain one entry per `<dataset>.parquet` available within the `parquet_folder`, as described in Subsection 4.4.2

The `wordlists_folder` serves a dual purpose. On the one hand, it functions as input to the IMT application, in the sense that it provides the lists created by the user outside the application. On the other hand, it is used to store the wordlists generated during the application execution. It will be composed of one json file per available wordlist. The wordlists must be located under this folder and follow the format described in Subsection 4.4.2.

4.4.2. Description of entities

Within the TMT, we work mainly with four entities, namely **local datasets**, **(logical) training datasets**, **wordlists**, and **topic models**, each of them managed by the modules related to the topic modeling back-end services above mentioned (i.e., `manageCorpus` for the local datasets and the training corpora; `manageLists` for the wordlists; and `topicmodeling`, for the topic models). Below we offer a more detailed description of these entities.

- *Local dataset*. It represents a dataset that has been downloaded from the data catalog. It is provided to the application through one of the external folders `parquet_folder` (see Subsection 4.4.1 for details) and all the information related to it must be enclosed within a json file named `datasetMeta.json`, which must contain a set of required fields (Table 9) for each local dataset, as exemplified below. As it can be seen, `datasetMeta.json` contains a dictionary of dictionaries, where each key indicates the path where each local dataset is available (relative to `parquet_folder`). Note that all the information related to

the local datasets must be provided within one unique json file (`datasetMeta.json`), but not one per dataset¹¹.

```
{
  "CORDIS.parquet": {
    "name": "CORDIS.parquet",
    "description": "All FP7 and H2020 projects in CORDIS",
    "visibility": "Public",
    "download_date": "2022-06-12 00:15:15.843834",
    "records": 61163,
    "source": "CORDIS",
    "schema": [
      "id",
      "title",
      "objective",
      "startDate",
      "ecMaxContribution",
      "euroSciVocCode",
      "rawtext",
      "lemmas"
    ]
  }
}
```

Table 9. Fields of the TMT entity “Local dataset”

Field	Description
name	Name of the local dataset
description	Description of the local dataset
visibility	Visibility level of the local datasets, i.e., ‘public’ (everyone has access to it) or ‘private’ (only the user that created the dataset has access to it)
download_date	The date at which the local dataset was downloaded
source	Name of the directory within the <code>parquet_folder</code> in which the dataset is contained.
schema	Columnar information saved for each element in the dataset in the parquet files

¹¹ The final format of the downloaded datasets and the json file associated with them must in any case be agreed during the integration of the TMT and the IMT with other IntelComp components, in particular with the data mediators that provide access to the available data sets.

- (Logical) training dataset. It represents a subset of one or several local datasets that can be used for training. At the time being, the TMT only supports the creation of training datasets for topic modeling, the functionality so all types of training datasets which must be supported by the ITM can be generated. All the information related to it is stored in a json file entitled with the same name the entity has. It must have a set of fields, as described in Table 10. Below, we show an example of this json file for a topic modeling training dataset.

```

{
  "name": "Cordis",
  "description": "Cordis training dataset for TM",
  "valid_for": "TM",
  "visibility": "private",
  "Dtsets": [
    {
      "parquet": "/Users/Documents/Intelcomp/fromHDFS/CORDIS.parquet",
      "source": "CORDIS",
      "idfld": "id",
      "lemmasfld": [
        "rawtext"
      ],
      "filter": ""
    }
  ],
  "creation_date": "2022-07-12 15:15:24.070089"
}

```

Table 10. Fields of the TMT entity “Training dataset”

Field	Description
name	Name of the training dataset
description	Description of the training dataset
valid_for	Purpose of the training dataset, e.g. ‘TM’ (topic modeling)
visibility	Visibility level of the training dataset, i.e., ‘public’ (everyone has access to it) or ‘private’ (only the user that created the dataset has access to it)
Dtsets	<p>A list describing the group of local data sets utilized for the generation of the training dataset. The dictionary describing each of the local datasets must have the following fields:</p> <ul style="list-style-type: none"> - parquet: Path to the local dataset - source: Name of the local dataset - lemmasfld: Field(s) of the local dataset used for the lemmas - filter: It can be provided by advanced users to filter the local dataset under some conditions. (currently not functional)

creation_date	Date at which the training dataset was created.
---------------	---

- Wordlists.** It defines a list of words that can be of multiple types, namely: stopwords, equivalent terms, keywords, or labels for the automatic topic labeling service. It can be created outside of or through the TMT (by means of one of the external folders **wordlists_folder**). In both cases, it must be saved in a json file following a fixed schema, as described in Table 11. Below, an example of a wordlist is provided¹²:

```

{
  "name": "english_generic",
  "description": "List of English stopwords",
  "valid_for": "stopwords",
  "visibility": "Public",
  "wordlist": [
    "a",
    "able",
    "about",
    "above",
    "according",
    "accordingly",
    "across",
    "actually",
  ],
  "creation_date": "2022-06-08 21:03:46.983949"
}

```

Table 11. Fields of the TMT entity “Wordlist”

Field	Description
name	Name of the wordlist
description	Description of the wordlist
valid_for	Type of list, which must be one of ‘stopwords’, ‘equivalences’, ‘keywords’ or ‘atl’
visibility	Visibility level of the wordlist, i.e., ‘public’ (everyone has access to it) or ‘private’ (only the user that created the wordlist has access to it)
wordlist	The actual content of the wordlist
creation_date	The date at which the wordlist was created

¹² For convenience, a small portion of the actual wordlist is shown.

- *Topic Model*. It represents a trained topic model. Again, all its information is saved in a json file named `trainconfig.json`, which is stored within the folder created for the training of the topic model in the project folder. The json file must contain the fields described in Table 12. Below, an example of a level-1 model trained with Mallet is given.

```
{
  "name": "Mallet-25",
  "description": "Mallet-25",
  "visibility": "Private",
  "trainer": "mallet",
  "TestSet": "/Users/Intelcomp/project_folder/datasets/Cordis.json",
  "Preproc": {
    "min_lemas": 15,
    "no_below": 10,
    "no_above": 0.6,
    "keep_n": 500000,
    "stopwords": [],
    "equivalences": []
  },
  "TMparam": {
    "mallet_path": "/Users/topicmodeler/mallet-2.0.8/bin/mallet",
    "ntopics": 25,
    "alpha": 5.0,
    "optimize_interval": 10,
    "num_threads": 4,
    "num_iterations": 1000,
    "doc_topic_thr": 0.0,
    "thetas_thr": 0.003,
    "token_regexp": "[\\p{L}\\p{N}][\\p{L}\\p{N}\\p{P}]*\\p{L}"
  },
  "creation_date": "2022-07-20 10:48:11.125048",
  "hierarchy-level": 0,
  "htm-version": null
}
```

Table 12. Fields of the TMT entity “Topic Model”

Field	Description
name	Name of the topic model
description	Description of the topic model
visibility	Visibility level of the topic model, i.e. ‘public’ (everyone has access to it) or ‘private’ (only the user that created the topic model has access to it)
trainer	Topic modeling algorithm used for training the topic model (‘mallet’, ‘sparkLDA’, ‘prodLDA’ or ‘ctm’).
TrDtSet	Path to the json file describing the training dataset
Preproc	Dictionary describing the preprocessing settings used
TMparam	Dictionary describing the training parameters, whose fields are dependent on the topic modeling algorithm used for training
creation_date	The date at which the model was created
hierarchy-level	0 or 1 describing whether the model is a level-1 or level-2 topic model
htm-version	Only different from null when the model is a level-2l topic model, it specifies the HTM algorithm used for the generation of the model’s training corpus

4.5. Command line API for Backend services support

In this section, the commands required to execute the back-end application services are defined. The application is divided into subsections that manage a certain part of the entire application and will be dockerized (i.e., one or more docker images will be created with the applications components) and executed separately. The source code for these components is provided in the following folders of the project:

- src/manageCorpus
- src/manageLists
- src/topicmodeling (two components)

Following WP5 indications, interaction with the back-end services will be based on the execution of independent commands on these dockers, i.e., each interaction will run a different command and context between commands will be lost. For this reason, project information will be shared in a common folder that will be mapped to each docker after initialization. Since context information will not be available, all necessary status information needs to be saved to these folders, using the kinds of entities and folder structure described in Subsection 4.4.

Another relevant issue is that of information exchange between the calling process and the back-end service. The preferred method for this will be through parameters provided in the command function. When this is not practical due to the complexity of the information that needs to be

exchanged, a JSON file will be exchanged using `STDIN` and/or `STDOUT`. In particular cases, it could also be considered to use temporary files written to a standard location. Currently the toolbox does not use such an approach for any of the services, but future updates will accommodate the needs for integration of the toolbox with other IntelComp tools.

Before moving forward to the description of the specific functionalities that are provided by the TMT components, we provide a general specification about how components will expose their functionality to other IntelComp components.

This definition depends on the inner organization of the application. It is probably best to use an entrypoint when creating the dockerfile so that the user selects only the specific arguments.

E.g.:

```
dockerfile: entrypoint ["python", "main.py"]
command: docker run image arg1 arg2
command effectively executed in app: python main.py arg1 arg2
```

Also, we need to add the necessary volumes.

Base:

```
docker run <--rm> -i <--name container_name> <-v /path/to/local:/path/to/container> image_name <args>
```

- `<container_name>`
Unique name for the container, can be a combination of username, date, etc.
- `image_name`
Image name generated in build
- `<args>`
List of arguments for each service

Flags:

- `--rm`
Remove the container when execution ends. (optional)
- `-i`
Set interactive mode. It is required to use standard input
- `--name`
A name for the container. (optional)
- `-v`
Volume binding. It maps a local directory to a directory inside the container so that local files can be accessed from it. The format is:
`/absolute/path/to/local/dir:/absolute/path/to/container/dir`

4.5.1. Corpus Management services (src/ManageCorpus)

Used to manage all corpus services.

image_name: mng-corpus

- List downloaded corpus: returns all available Datasets in HDFS.
--listDownloaded: List datasets downloaded from HDFS with metadata.
- List logical corpus: returns all available logical Datasets for training.
--listTrDtsets: List Training Datasets
- Add corpus: creates a new logical Dataset from a json description.
--saveTrDtset: Create and save Training Dataset
Currently, a new word list is created using the standard input, thus adding all elements manually in the terminal. In future updates, a file can be passed as an argument where all the required information is previously stored.
- Remove corpus: deletes a logical Dataset
--deleteTrDtset <corpusname>: Delete a Training Dataset
- Rename corpus
--rename <corpusname> <new_name>: Rename a Training Dataset.
- Copy corpus
--copyCorpus <corpusname>: Make a copy of a corpus with the name *corpusname-copy*.

Examples:

Execute help. Help is the default option when executing the container, as specified in dockerfile

```
docker run --rm -i --name cnt-ja-mc -v  
/Users/josea/datasets/logical:/data/logical mng-corpus
```

List of downloaded corpus

```
docker run --rm -i --name cnt-ja-mc -v  
/Users/josea/datasets/logical:/data/logical mng-corpus --  
listDownloaded
```

List of logical corpus

```
docker run --rm -i --name cnt-ja-mc -v  
/Users/josea/datasets/logical:/data/logical mng-corpus --  
listTrDtsets
```

Create a new logical corpus

```
docker run --rm -i --name cnt-ja-mc -v
/Users/josea/datasets/logical:/data/logical mng-corpus --
saveTrDtset
```

Delete logical corpus

```
docker run --rm -i --name cnt-ja-mc -v
/Users/josea/datasets/logical:/data/logical mng-corpus --
deleteTrDtset test
```

Rename logical corpus

```
docker run --rm -i --name cnt-ja-mc -v
/Users/josea/datasets/logical:/data/logical mng-corpus --
renameTrDtset test test_renamed
```

Copy logical corpus

```
docker run --rm -i --name cnt-ja-mc -v
/Users/josea/datasets/logical:/data/logical mng-corpus --
copyTrDtset test
```

4.5.2. List Management services (src/ManageLists)

Used to manage all list management services.

image-name: mng-lsts

- List wordlists
--listWordLists: List available WordLists
- Add wordlist
--createWordList: Save wordlist
Currently, a new word list is created using the standard input, thus adding all elements manually in the terminal.
- Remove wordlist
--deleteWordList <wordlistname>: Delete *wordlistname* element.
- Rename wordlist
--renameWordList <wordlistname> <new_name>: Change old *wordlistname* to *new_name*.
- Copy wordlist
--copyWordList <wordlistname>: Make a copy of the element with name *wordlistname-copy*.

Examples:

Execute help. Help is the default option when executing the container, as specified in dockerfile


```
docker run --rm -i --name cnt-ja-ml -v
/Users/joseantem/Documents/projects/dock-
TM/ManageLists/wordlists:/data/wordlists mng-lsts
```

List of wordlists

```
docker run --rm -i --name cnt-ja-ml -v
/Users/joseantem/Documents/projects/dock-
TM/ManageLists/wordlists:/data/wordlists mng-lsts --
listWordLists
```

Create a new word list

```
docker run --rm -i --name cnt-ja-ml -v
/Users/joseantem/Documents/projects/dock-
TM/ManageLists/wordlists:/data/wordlists mng-lsts --
createWordList
```

Delete wordlist

```
docker run --rm -i --name cnt-ja-ml -v
/Users/joseantem/Documents/projects/dock-
TM/ManageLists/wordlists:/data/wordlists mng-lsts --
deleteWordList test
```

Copy wordlist

```
docker run --rm -i --name cnt-ja-ml -v
/Users/joseantem/Documents/github/topicmodeler/wordlists:/data/w
ordlists mng-lsts --copyWordList test
```

4.5.3. *Topic Modeling services (src/topicmodeling)*

Used to manage all topic modeling services. Contain two 'managers'

topicmodeling:

Manages the preprocessing, generation and training of a topic model.

image-name: tmt-mdls

- Create model
 - spark: whether spark is available.
 - preproc: whether to preprocess training data.
 - train <configfile>: train topic model according to configuration file.
 - hierarchical <configfilechild>: create submodel according to 'child' configuration file

Examples:

Create a new model

```
docker run --rm -i --name cnt-ja-tm -v
/Users/joseantem/Documents/models:/models tmt-mdls --preproc --
train /models/mallet-30t-ai/config.json
```

manageModels:

Topic model manager, similarly to corpus and wordlists managers, allows the removal, renaming, copying, etc. of topic models.

image-name: mng-mdls

- List models
--listTMmodels: List available models
- Remove model
--deleteTMmodel <modelname>: Delete *modelname* element.
- Rename model
--renameTM <modelname> <new_name>: Change old *modelname* to *new_name*.
- Copy model
--copyTM <modelname>: Make a copy of the element with name *modelname-copy*.
- Show model topics
--showTopics <modelname>: retrieve topic labels and word composition for selected model.
- Set topic labels
--setTpcLabels <modelname>: set topic labels for selected model.
- Remove topics
--deleteTopics <modelname>: removes topics from selected model.
- Sort topics
--sortTopics <modelname>: sort topics according to size.
- Reset model
--resetTM <modelname>: reset model to its initial state.

Examples:

Execute help. Help is the default option when executing the container, as specified in dockerfile

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls
```

List of models

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --
listTMmodels
```

Create a new model.

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --createModel
```

Delete model

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --
deleteTMmodel test
```

Rename model

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --renameTM
test
```

Copy model

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --copyTM test
```

Show model topics

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --showTopics
test
```

Set topic labels

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --
setTpcLabels test
```

Remove topics

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --
deleteTopics test
```

Sort topics

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --sortTopics
test
```

Reset model

```
docker run --rm -i --name cnt-ja-mm -v
/Users/joseantem/Documents/models:/models mng-mdls --resetTM
test
```

4.5.4. Deployment flow

An example of the actual development of the aforementioned services is described below, in particular the generation of a topic model as the wordlist management is straightforward and the corpus management follows a similar schema.

1. Generate docker services

In some cases, such as wordlist management, only a container is required to run at the same time, as the operations are not time-consuming. On the other hand, the creation of models may take hours or days to complete, thus dedicated containers will be created.

2. Create a model

The first step will always be generating a configuration file that contains all the desired model parameters. This is an example of a standard *config.json* file to create a mallet topic model:

```
{
  "trainer": "mallet",
  "TMparam": {
    "ntopics": 50,
    "thetas_thr": 0.003,
    "labels": "",
    "mallet_path": "/home/joseantem/mallet-2.0.8/bin/mallet",
    "alpha": 5.0,
    "optimize_interval": 10,
    "num_threads": 8,
    "num_iterations": 500,
    "doc_topic_thr": 0.0,
    "token_regexp": "[\\p{L}\\p{N}][\\p{L}\\p{N}\\p{P}]*\\p{L}"
  }
}
```

The entire folder structure of the model will be constructed taking this file as base.

3. Model training

For the training, a corpus must be selected and optionally preprocessed and the training process will automatically begin with the *create model* command from **topicmodeling**. The training process includes:

- a. Creation of an adjusted model usable for inference.
- b. Coherence measures.
- c. Automatic labeling of the topics.
- d. Other model files such as alphas, betas, thetas or vocabulary.

In this case there is no final output, just the files created.

4. Tuning

In case it is necessary, the model can be tuned following any of the other command options.

The final structure of the model directory will result in something similar to this:

```
.
├── TModel
│   ├── alphas.npy
│   ├── alphas_orig.npy
│   ├── betas.npy
│   ├── betas_ds.npy
│   ├── betas_orig.npy
│   ├── d3.js
│   ├── edits.txt
│   ├── ldavis.v3.0.0.js
│   ├── ndocs_active.npy
│   ├── pyLDAvis.html
│   ├── thetas.npz
│   ├── thetas_orig.npz
│   ├── topic_coherence.npy
│   ├── topic_entropy.npy
│   ├── tpc_descriptions.txt
│   ├── tpc_labels.txt
│   └── vocab.txt
├── config.json
├── corpus.txt
├── modelFiles
│   ├── corpus.mallet
│   ├── diagnostics.xml
│   ├── doc-topics.txt
│   ├── inferencer.mallet
│   ├── mallet.config
│   ├── topic-report.xml
│   ├── topickeys.txt
│   └── word-topic-counts.txt
├── stats
│   ├── mem_use.txt
│   └── tuned_params.json
```

4.6. TMT Front Ends

As already mentioned, the toolbox provides two user interfaces, one based on interaction with the user through a terminal, and the second based on a GUI. These tools will be useful both to IntelComp operators (during the period where the IMT is not available for use), as well as to the IMT front-end developers.

In this subsection we describe the operation of both UIs. The functionalities of both UIs are equivalent except for the availability of an additional functionality in the terminal-based UI to copy data from the local HDFS infrastructure at UC3M to folder `parquet_folder`; this functionality was only necessary for development purposes, as it will be replaced by IntelComp data mediators during integration in the IMT. Apart from this, the only difference between the terminal-based and graphical UIs is how the information is presented to and input is taken from the user.

4.6.1. Execution commands

In order for the user to start any version of the application, the following command needs to be executed:

```
python main_script --p project_folder --parquet parquet_folder
--wlist wordlists_folder
```

where:

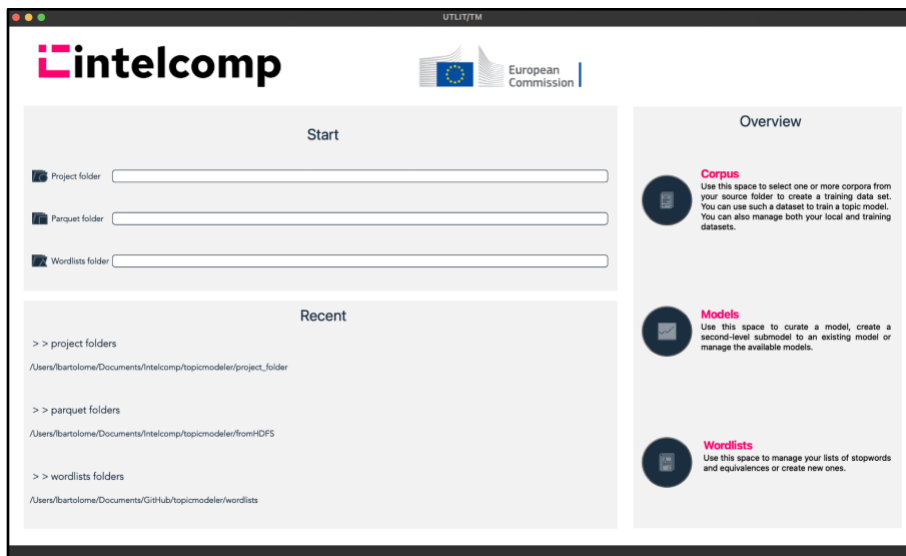
- **main_script** refers to the script that relates to the version of the application to be executed. Use
 - **ITMT_mainCMD.py** for the command line user interface,
 - **ITMT_mainGUI.py**, for the graphical user interface.
- **project_folder** is the path to a new or an existing project in which the application's output will be saved.
- **parquet_folder** is the path to the downloaded parquet datasets.
- **wordlists_folder** is the path to the folder in which the wordlists are saved.

Note that for the case of the graphical user interface, the application can also be invoked without parameters, being possible to select them from the application front page as follows:

```
python ITMT_mainGUI.py
```

If the application is invoked with parameters, the project folder, parquet folder and wordlists folder paths are written on their respective text boxes (i.e., white spaces located at the right of each selection button), as shown in Figure 10, being possible to change the selection by clicking on their respective associated button.

Figure 10. Graphical user interface; The welcome page



Given that the application is invoked without parameters, the text boxes are shown empty. In each of the three options, the user's file system is popped-up so that she/he can easily search for the project/parquet/wordlists folders. Alternatively, if it is not the first time that the graphical user interface is initiated, the user can select the project/parquet/wordlists folder from the "Recent" folders shown in the bottom of the application.

For the case of the command-line application, its main menu is shown immediately after the invocation of the execution command. As for the graphical user interface, once the input folders have been selected, the menu buttons are unlocked, and the user can proceed with any of the GUI's functionalities.

4.6.2. *Command line user interface*

This user interface is based on a menu, as shown in Figure 11. The user can navigate through the application by writing the number associated with each functionality. When the application needs input from the user, she/he will be requested to do so.

Figure 11. Command line user interface main menu

```

*****
*** MAIN MENU.
Available options:
  1. Corpus Management Options
  2. WordLists Management Options
  3. Topic Modeling options
  0. Exit the application

What would you like to do? [0-3]:
  
```

As the user interacts with the application, new options will be shown. A configuration file is available in /config/ITMTmenu.yaml, defining the logic of operation of the console-based interface.

4.6.3. *Graphical user interface*

The graphical user interface is composed of four main subwindows, each of them relating to one of the distinct functionalities the application has, in addition to one more subwindow for the welcome page. The latter can be accessed by their corresponding buttons on the left menu. Below we describe thoroughly each functionality/subwindow.

Corpus Management subwindow

As shown in Figure 12 and Figure 14, two different views are available: one visualizing and operating the available local datasets (obtained from the parquet folder), and a second one for the same functions but for the training datasets.

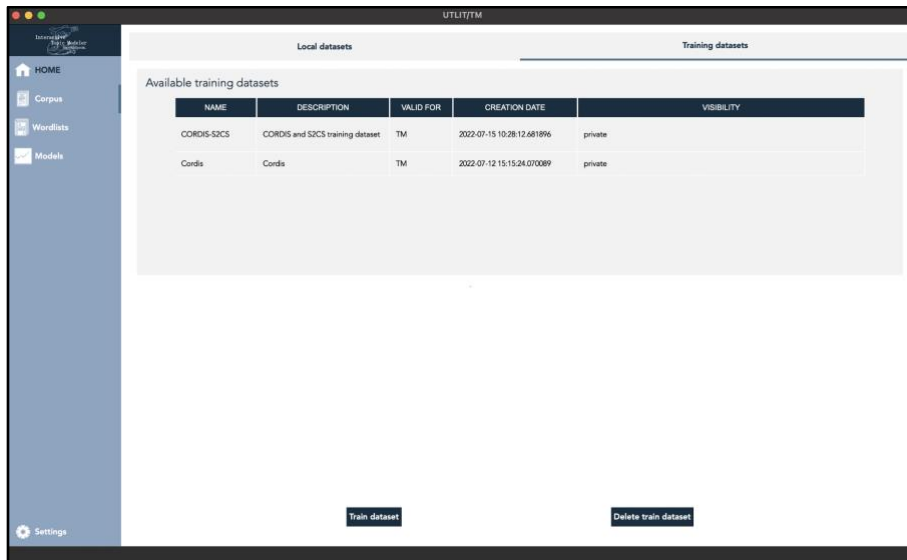
Figure 12. Available local datasets view

ADD TO TRAIN?	NAME	SOURCE	DESCRIPTION	NUMBER OF DOC	DOWNLOAD DAT	VISIBILITY	FIELDS
<input checked="" type="checkbox"/>	CORDIS.parquet	CORDIS	All FP7 and H2020 projects in CORDIS	61163	2022-06-12 00:15:15.843...	Public	id, title, objective, startDate, eCMarContribution, euroSciVocCode, rawtext, lemmas
<input checked="" type="checkbox"/>	S2CS.parquet	Semantic Scholar	Computer Science papers in Semantic Scholar	14801878	2022-06-12 00:36:16.109...	Public	id, title, paperAbstract, doi, year, fieldOfStudy, rawtext, lemmas
<input type="checkbox"/>	S2Bo.parquet	Semantic Scholar	Biology papers in Semantic Scholar	14602379	2022-06-12 01:04:02.131...	Public	id, title, paperAbstract, doi, year, fieldOfStudy, rawtext, lemmas
<input type="checkbox"/>	S2Med.parquet	Semantic Scholar	Medicine papers in Semantic Scholar	49114786	2022-06-12 01:28:30.736...	Public	id, title, paperAbstract, doi, year, fieldOfStudy, rawtext, lemmas

From the first view, we can create a new training dataset by selecting the checkboxes of the downloaded datasets that we want to use for generating the training dataset and clicking the “Generate train dataset button”. This action leads to a new window (Figure 13), from which the user can select the characteristics of each of the selected datasets that are going to be used for the training dataset generation; these include, for each selected dataset: 1) the fields to be used for the raw text; 2) the fields to be used for the lemmas; 3) additional filter conditions that can be specified from the “Advanced filter options” tab. Additionally, the id to be used as an identifier, the privacy level, name and description of the new training dataset must be specified. Once all the fields have been filled out, the user can click the “Create training dataset” button, which leads to the creation of the new training dataset.

Figure 13. Training dataset generation window



Figure 14. Available training datasets view



Regarding the second view of the corpus management subwindow, it allows the user to visualize the available training datasets, as well as delete any of them. Additionally, this view leads to the topic modeling training functionalities: by clicking on the dataset that is desired to be trained on (i.e., the “Train dataset” button), the user is redirected to a sequence of two new windows (Figure 15 and Figure 16) that allows the preprocessing and training of the selected dataset.

Preprocessing and topic modeling training windows

From the preprocessing window (Figure 15) the user can select the settings that are going to be used for the preprocessing of the selected training dataset, namely the minimum number of lemmas that a document must have to be kept in the final training corpus (`min_lemmas`); the maximum and the minimum number of occurrences (`no_above` and `n_below`) according to which the words are removed (words with too high frequencies, infrequent words or very rare); and the maximum number of words to be kept in the vocabulary (`keep_n`). Additionally, by navigating to Stopwords and Equivalences tabs, the user can select one or more lists of stopwords and equivalences to be used to filter the corpus.

Once the preprocessing parameters have been specified by the user, the training window (Figure 16) immediately pops up. From this window, the user can select which topic modeling technique (LDA-Mallet, Spark-LDA, AVITM, or CTM) is going to be used for the training of the previously selected training corpus (buttons located at the top of the window). The parameters shown in the middle, left panel in the ‘Settings’ and ‘Advanced Settings’ will be updated according to the selected algorithm, and will be initialized with the default values. The user can update these values, but at the time the “Train” button is clicked, parameter checking will be carried out to ensure that the selected values meet the requirements necessary for each parameter. In case any of the parameters do not have a correct value, the symbol  will be shown next to it; otherwise, the symbol  will appear.

Additionally, the middle, right panel displays the logs of the training of the models and the bottom options allow the user to specify the name and description of the to-be-trained topic model, as well as its visibility level (public or private).

Figure 15. Topic modeling preprocessing window

Figure 16. Topic modeling training window

Wordlists management subwindow

Wordlists functionalities are held in the subwindow shown in Figure 17. Apart from the wordlist visualization, the user can edit or delete an existing wordlist, or create a new one, by either selecting the to be modified/deleted wordlist and the corresponding button, or by directly selecting the “Create new wordlist” button.

In both cases, creating or editing a wordlist, a new window is opened. The new wordlist creation subwindows (Figure 18(a)) facilitate the user in the determination of 1) the wordlist type; 2) its privacy level; 3) its name; 4) the wordlist content. The window for editing an existing wordlist (Figure 18(b)) is equivalent to the one just presented, with the only difference being that 1), 2)

and 3) are not editable, and the current content of the under edition wordlist is shown and can be modified as desired.

Figure 17. Wordlists management subwindow

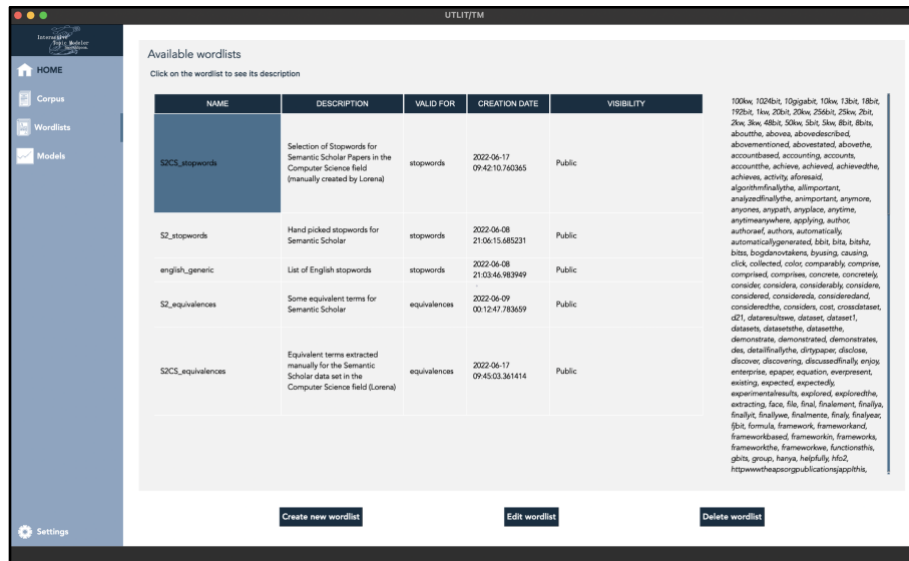
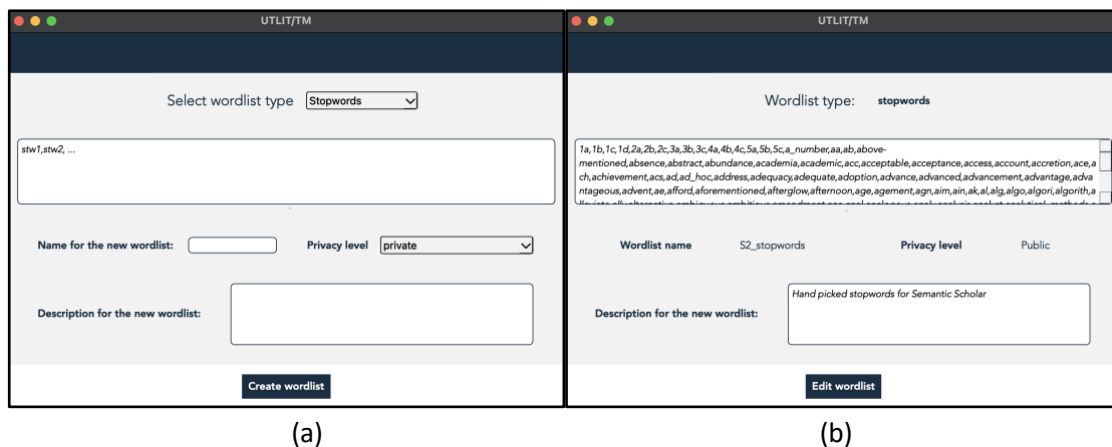


Figure 18. Wordlists creation (left) / edition (right) subwindows



(a)

(b)

Models' management subwindow

All model-related functionalities are accessible from the Models management subwindow, as shown in Figure 19.

At the top, we can see a list in which all the topic models trained for the project in use are listed; note that if 2-level topic models have been generated, they will be gradually listed. By clicking on one of these models, the middle table shows the information related to the model selected (i.e., name, description, visibility, trainer, training dataset, hierarchical level, and type and creation date) as well as the information related to each of its topics (id, size, label, number of active documents and description) shown in the table at the bottom of the page.

Figure 19. Model management subwindow

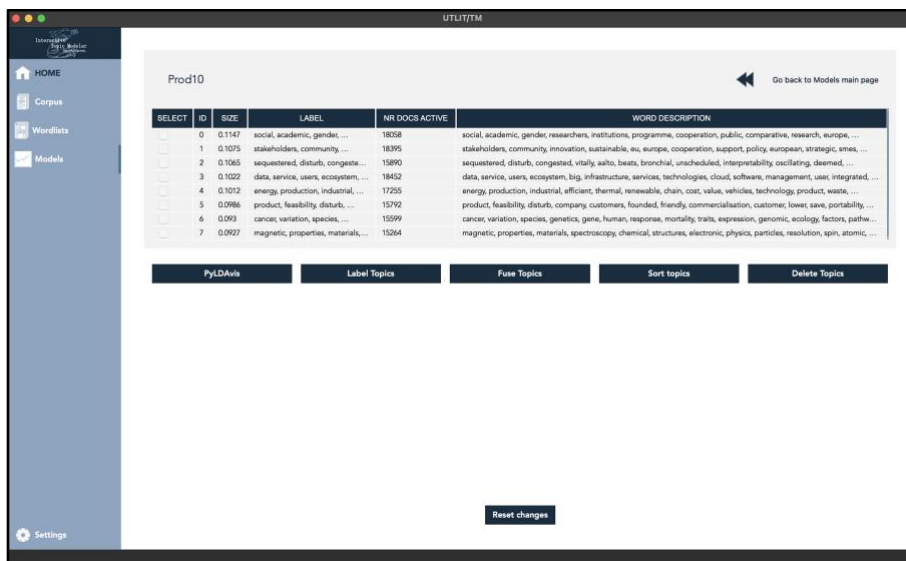
The screenshot shows a web interface for model management. At the top, it says 'Available trained models'. Below this, there are two rows of model information: 'Prod10' and 'Prod25'. A table below lists model details with columns: NAME, DESCRIPTION, VISIBILITY, TRAINER, TRAINING DATASET, HIERARCHY LEVEL, HIERARCHICAL TYPE, and CREATION DATE. Below the table are four buttons: 'Make a copy', 'Rename model', 'Curate model', and 'Delete model'. Below these buttons is a section titled 'Topics' information of the selected model', with a sub-instruction: 'Click on one of the topics and the "Train submodel" button to create a 2-level submodel'. This section contains a table with columns: ID, SIZE, LABEL, NR DOCS ACTIVE, and WORD DESCRIPTION. Below this table is a 'Train submodel' button.

NAME	DESCRIPTION	VISIBILITY	TRAINER	TRAINING DATASET	HIERARCHY LEVEL	HIERARCHICAL TYPE	CREATION DATE
Prod10	Prod10	private	prod.DA	/Users/fbarloma/Documents/intelcomp/topicmodeler/project_folder/...	0	Name	2022-07-18 13:10:03.505433

ID	SIZE	LABEL	NR DOCS ACTIVE	WORD DESCRIPTION
0	0.1147	social, academic, gender, ...	18058	social, academic, gender, researchers, institutions, programme, cooperation, public, comparative, research, europe, education, national, internation...
1	0.1075	stakeholders, community, ...	18395	stakeholders, community, innovation, sustainable, eu, europe, cooperation, support, policy, european, strategic, smes, projects, network, needs
2	0.1065	sequestered, disturb, ...	15890	sequestered, disturb, congested, vitality, aalto, beats, bronchial, unscheduled, interpretability, oscillating, deemed, standpoint, portability, 1996, ...
3	0.1022	data, service, users, ...	18452	data, service, users, ecosystem, big, infrastructure, services, technologies, cloud, software, management, usec, integrated, platform, end
4	0.1012	energy, production, industri...	17255	energy, production, industrial, efficient, thermal, renewable, chain, cost, value, vehicles, technology, product, waste, products, technologies
5	0.0986	product, feasibility, disturb, ...	15792	product, feasibility, disturb, company, customers, founded, friendly, commercialisation, customer, lower, save, portability, patented, equipment, ...
6	0.093	cancer, variation, species, ...	15399	cancer, variation, species, genetics, gene, human, response, mortality, traits, expression, genomic, ecology, factors, pathway, bacterial
7	0.0927	magnetic, properties, ...	15284	magnetic, properties, materials, spectroscopy, chemical, structures, electronic, physics, particles, resolution, spin, atomic, atoms, optical, synthesis
8	0.0921	cancer, cellular, function, ...	15531	cancer, cellular, function, therapies, biology, mechanisms, molecular, expression, gene, pathway, response, binding, identify, epigenetic, dna
9	0.0914	optical, techniques, ...	16564	optical, techniques, computational, processing, applications, resolution, methods, devices, detection, imaging, quantum, properties, engineering, ...

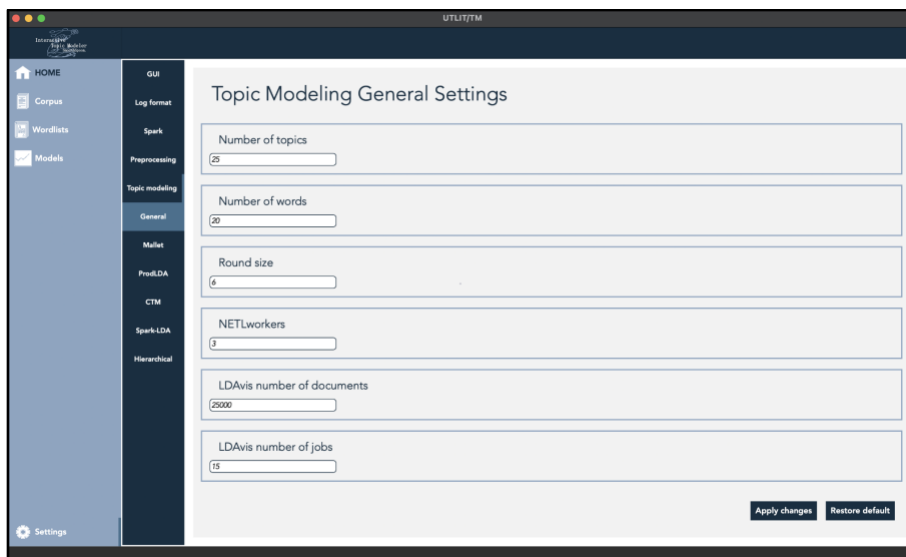
In the middle of the view, we can distinguish a row of 4 buttons that allow the user to create a copy of, rename or delete a topic model. Additionally, the button “Curate model” allows the user to access a new window view with the curation tools (see Subsection 3.4.2 for a detailed description of all available curation tools), as shown in Figure 20. On the top of this view, we can visualize the same topics’ information of the selected model for curation, as presented above; though, in this case, the table serves two additional purposes: 1) the user can manually modify the content of each row (topic) of the column “Label” and subsequently click the button “Label Topics” to manually label the topics; 2) the user can select as many topics by clicking on the checkboxes located at the first column on the table and subsequently click on one of the buttons “Fuse topics” or “Delete topics” to fuse and/or delete topics, respectively. Additionally, topics can be sorted by clicking on “Sort topics”; the PyLDAvis graph can be visualized by clicking on the PyLDAvis button; the model can be reset to its default value before curation by clicking on the “Reset model” button.

Lastly, clicking on one of the topics related to the selected model and the button at the bottom with the name “Train submodel” will lead to the opening of the training window Figure 16), but with two additional parameters to select: 1) the hierarchical topic model version that is going to be used for the creation of the 2-level topic model’s training corpus (‘htm-ds’ or ‘htm-ws’), and 2) if ‘htm-ds’ is selected as HTM version, a value for the parameter “threshold” must be provided as well. Once these two parameters have been selected, the training of the submodels proceeds in the same way the training of a 1-level model does, being again possible for the user to select the topic modeling algorithm for training and configure its parameters; in this case, though, no preprocessing is carried out.

Figure 20. Curation subwindow


Settings subwindow

From this view (Figure 21), the user can manage all the configuration settings implied in the TMT tool, logging, and Spark settings, as well as those directly related to the preprocessing pipeline and the topic modeling algorithms. Fundamentally, it allows the user to modify the parameters given in the configuration file, as specified in Subsection 4.3.

Figure 21. Settings subwindow


5. PERFORMANCE EVALUATION

In this section, we report on a series of experiments regarding the evaluation of the performance of the different techniques implemented in the TMT toolbox, in order to understand their

properties in terms of scalability and quality of the topics obtained, thus providing information that can guide the selection of one or another tool for different situations and data sets.

In Section 5.1, we evaluate the TMT data preprocessing workflow and the alternative options/configurations offered by the TMT toolbox. To this end, a set of experiments were conducted with the focus on scalability. In the next section, we evaluate topic models by employing two well-known coherence measures as proxies to the quality of the topics provided by the different TMT models.

5.1. Data Preprocessing

The TMT toolbox offers two possibilities to parallelize the preparation of the training dataset by exploiting two different distributed frameworks: the use of the Dask framework on a powerful machine (physical or virtual) or parallelization over a Spark cluster. Given the differences of the two solutions at the infrastructure level, it is not possible to make a direct comparison between them, so their properties in terms of scalability will be analyzed independently. Nevertheless, the required computation time and the characteristics of the infrastructure on which the experiments have been carried out will allow drawing some conclusions regarding the preference for one or another solution.

5.1.1. Parallelization by employing the Dask framework

To begin with, we have carried out the analysis of the execution times for the preprocessing of a small training dataset. This dataset consists of the summaries of the projects available on CORDIS (FP7 and H2020) that represent a total of 60,596 documents.

Data preprocessing in the TMT implementation with Dask consists of the following steps:

- T1. Text homogenization, elimination of specific stopwords, and application of word replacements (i.e., equivalences),
- T2. Creation of a dictionary and filtering of terms of little or excessive use,
- T3. Export of the training dataset

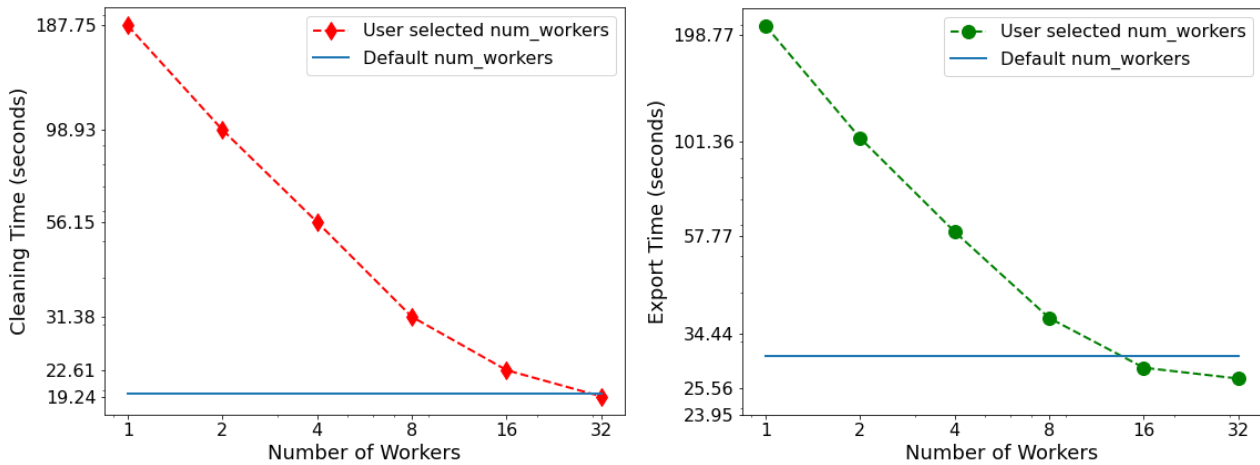
The first and third steps have been parallelized with Dask, while the second step was carried out in a single thread with Gensim.

To analyze the scalability of the provided implementation, experiments have been carried out selecting a variable number of workers. This first set of experiments was run on a server with 40 Intel Xeon cores @ 2.4 GHz, each with two threads, and a total RAM of 128 GB.

Figure 22 shows the time consumed for the homogenization and cleaning phase (left) and for the export of the data set to the format required by Mallet (right). For reference, the time required for Gensim dictionary generation in this specific case was just around 8 seconds. Our analysis shows that parallelization with Dask allows the processing time to be reduced approximately proportionally to the number of workers employed, although when using more than 16 or 32 workers the improvement flattens, and the times are saturated, even becoming slightly higher when using Dask default setting (i.e., number of available nodes, 40 in this case)

which is depicted with a blue horizontal line. Therefore, in this case it seems advisable to adjust the number of workers to approximately half the available cores in order to optimize performance vs. resource consumption.

Figure 22. Cleaning and export times for the CORDIS dataset using Dask with a variable number of workers



Scalability analysis for varying data set size

To evaluate the scalability of preprocessing with Dask at a larger scale, we have carried out processing time measurements working on a second training dataset consisting of all articles in the field of "Computer Science" present in the January 2022 edition of the Semantic Scholar dataset (hereinafter the S2CS dataset). This dataset is made up of a total of 14,801,878 documents; to evaluate the scalability we have also used random samples of the S2CS dataset keeping 30%, 10%, 3% and 1% of its documents.

The machine used in these simulations has 48 Intel(R) Xeon(R) Silver cores at 2.4 GHz, with two execution threads per core, and 128 GB of RAM available on the server.

Figure 23 shows the computational resources required (number of CPUs used, RAM memory and virtual memory) when processing the S2CS dataset with a 10% sampling, while using 4, 8, 16 and 32 workers for parallelization with Dask. In this case, we show resource consumption as a function of time. Our analysis can be summarized in the following points:

- All figures show a similar pattern consisting of the concatenation of tasks T1, T2, T3. Gensim dictionary generation (T2) is defined by the interval where CPU usage is reduced to one, whereas T1 and T3 occur before and after this phase and are characterized by using as many CPUs as made available for the simulation.
- The homogenization and cleaning phase (T1), as well as the export of the processed dataset (T3), reduce their duration inversely proportional to the number of cores used.
- Gensim dictionary generation uses a single server core in any case, and its duration is practically independent of the number of workers employed. Its relative duration is obviously greater when the number of workers employed grows, due to the shorter duration of the other phases that benefit from parallelization.

- RAM memory consumption is practically not affected by the number of workers used, although the virtual memory used does grow with the number of workers.

Figure 23. Memory and CPU usage for preprocessing and corpus preparation tasks using Dask with a variable number of workers. Dataset S2CS sampled at 10% (1,480,804 documents)

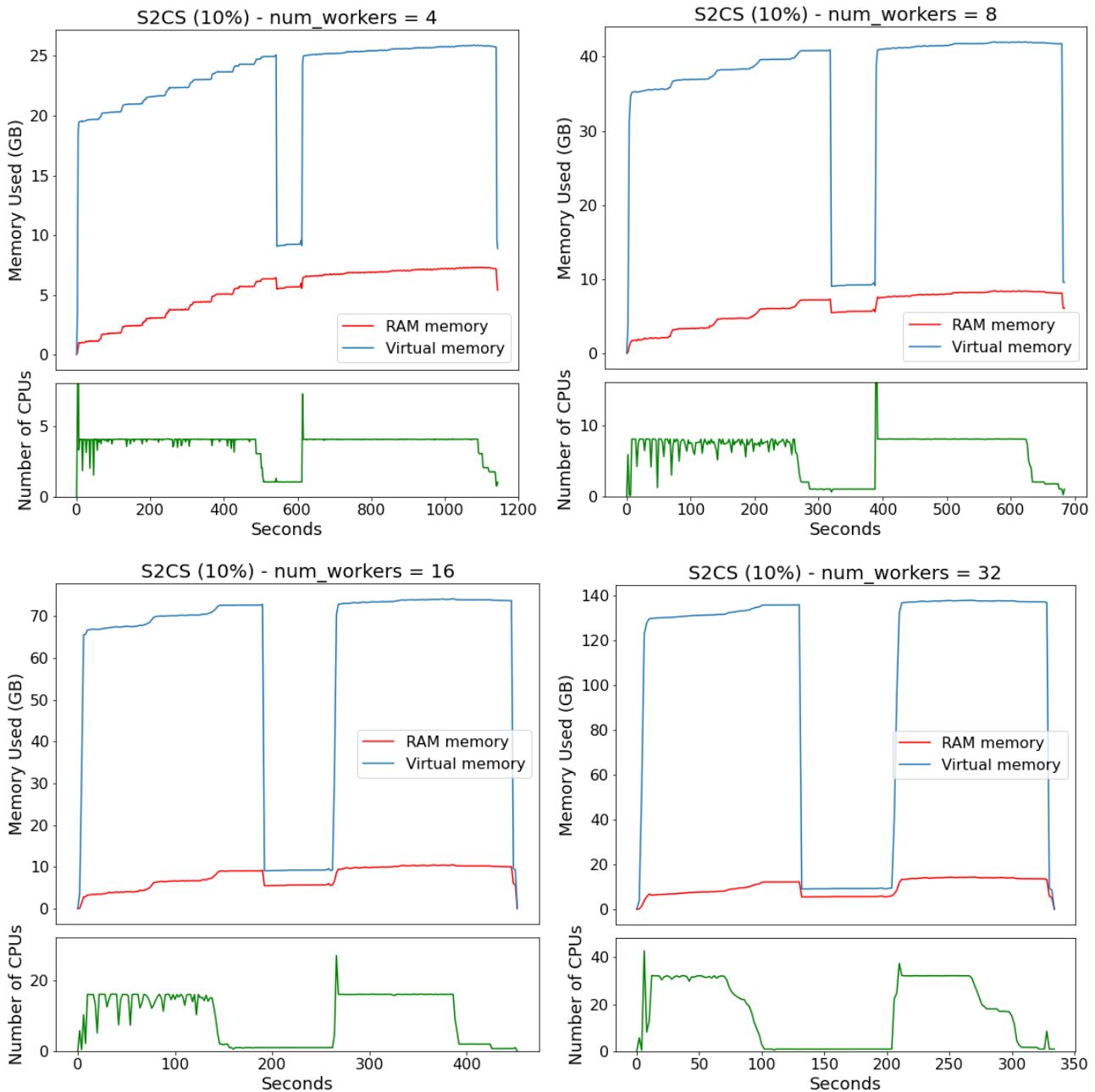


Figure 24 shows the same information when processing the entire S2CS dataset. The conclusions are similar to those already raised, observing the following main differences:

- Although memory consumption again remains relatively independent of the number of workers employed, it is approximately 10 times greater than that observed in Figure 23, because the processed dataset size has also grown in that proportion.

- Gensim dictionary generation phase seems to have a longer relative duration. This suggests that the duration of this second step grows more than linearly as the size of the dataset increases. In order to better appreciate this, Figure 25 shows the dictionary generation time by varying the size of the dataset. We can see how when going from 10% to 30%, and from 30% to 100%, the vocabulary generation time grows by a factor greater than 3 (note the logarithmic scale of the x-axis). The results in the figure also confirm our previous statement that the dictionary generation time is practically independent of the number of workers employed.

Figure 24. Memory and CPU usage for preprocessing and corpus preparation tasks using Dask with a variable number of workers. Dataset S2CS (14,801,878 documents)

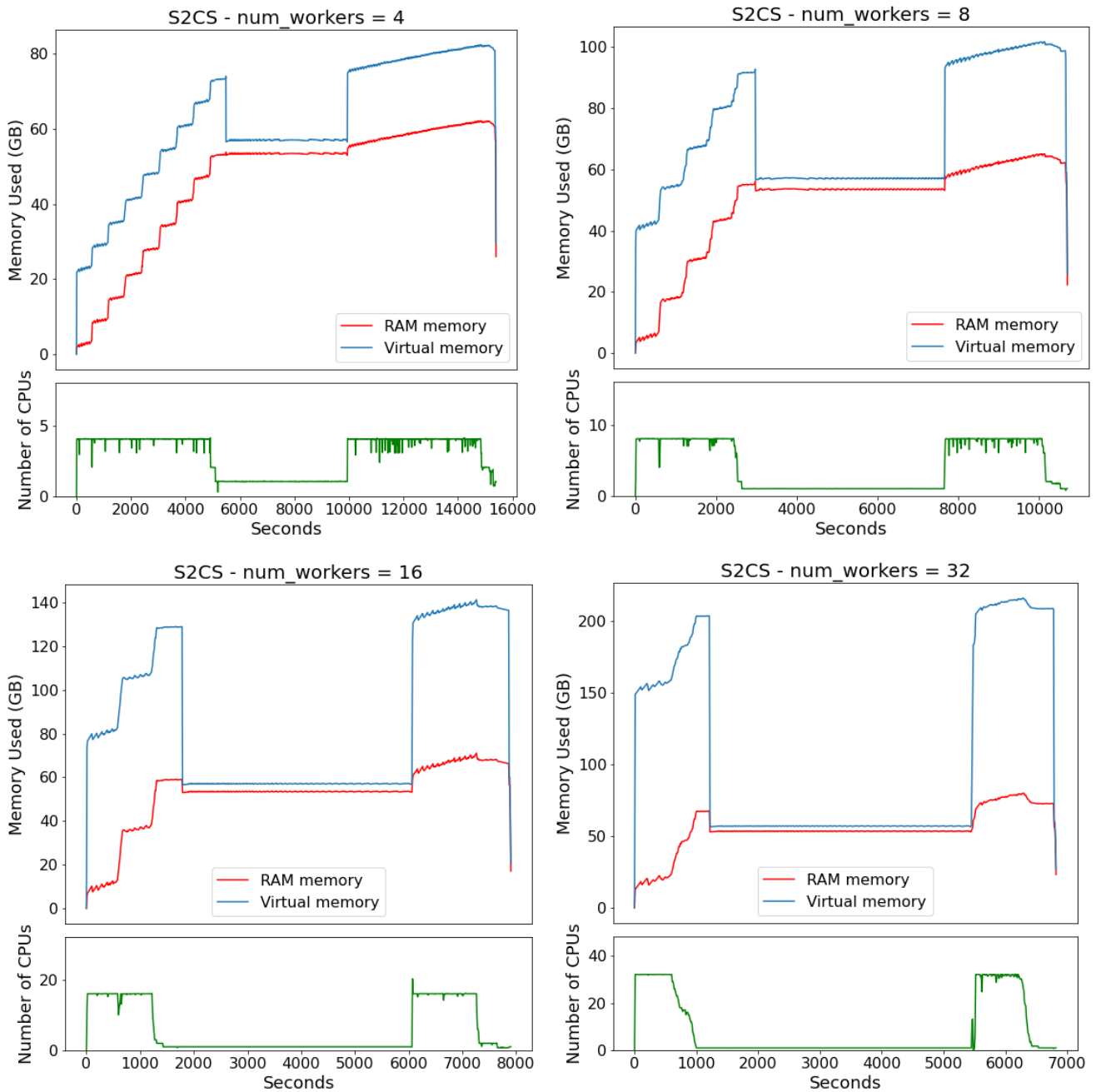
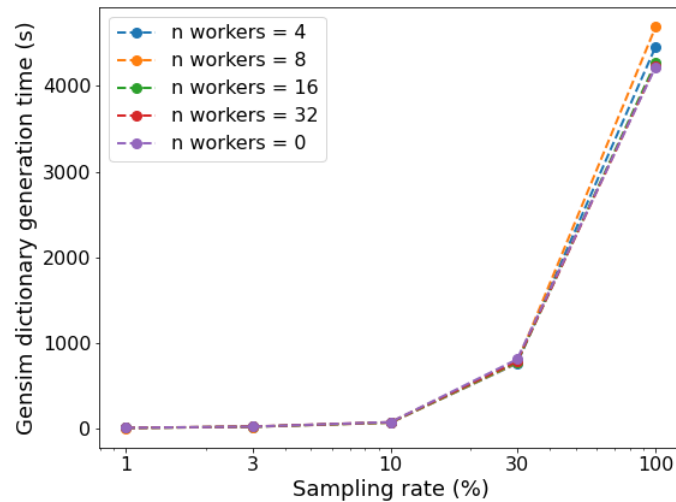


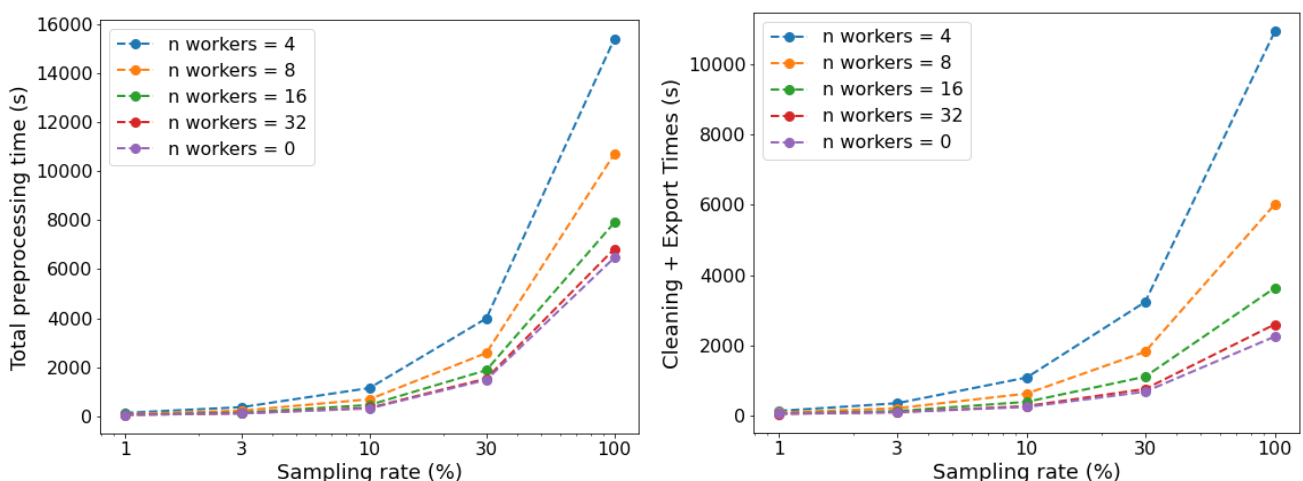
Figure 25. Time for generating the Gensim dictionary for the S2CS dataset as a function of the number of documents (varying sampling rate)



We conclude this subsection showing in Figure 26 the total time required for preparing the training datasets when using the Dask implementation. For the largest dataset up to around 16,000 seconds are needed, which can be reduced to roughly 6,000 seconds if using at least 16 workers. Approximately 65% of this time is consumed by the Gensim dictionary generation process, this being therefore the most demanding process, thus, the stumbling block in terms of efficiency, when processing large datasets.

All things considered, the Dask implementation allows for very important processing time savings for small and moderate size datasets (up to 3-5 million documents), but for larger size datasets the computation time savings are not that significant in relative terms since in this case the Gensim dictionary generation process becomes the most decisive limiting factor.

Figure 26. Total time for the S2CS dataset preprocessing (left) and total time just for the parallelized tasks as a function of the number of documents (varying sampling rate)



5.1.2. Parallelization by employing the Spark Framework

The second implementation available for data preprocessing involves three phases equivalent to those used in the case of Dask:

- T1. Text homogenization, elimination of specific stopwords, and application of word replacements (equivalences),
- T2. Creation of a Spark *CountVectorizer* object that allows TMT users/operators to select the words in the dictionary and obtain the vector representation of the documents,
- T3. Export of the training dataset.

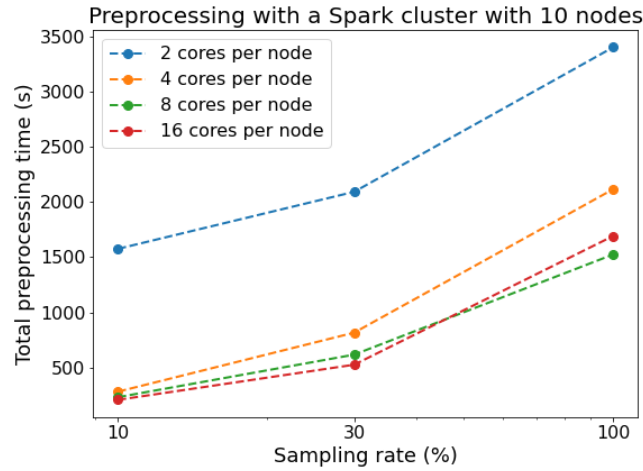
In this case, the three steps can benefit from the parallelization offered by having several processing cores, although the second of the steps listed involves the exchange of information between the nodes of the cluster, which in principle could even slow down the processing.

The exhaustive measurement of computational resources is not possible in the infrastructure available at UC3M, so only the total execution times have been measured. Even in this case, there may be non-negligible differences between executions, since the typology of cluster nodes may vary between them. Still, the results are clear enough to draw some useful conclusions regarding the efficiency of this implementation.

Figure 27 shows the execution times when processing the S2CS dataset using different sampling factors (full dataset, 30% and 10%), and using a variable number of cores in each of the 10 execution nodes. The results show:

- Shorter execution times are required for preprocessing as compared to Dask implementation, e.g., when using 4 cores per node (40 cores in total) the processing time was around 2,000 s (a reduction greater than 70% with respect to Dask)
- This implementation scales sublinearly: increasing the size of the training dataset by a factor of 10 implies an increase in processing time of less than that factor. This is probably due to a more efficient implementation of the dictionary computation task, although a greater benefit of the dataframe parallelization process should also be considered as the size of the training set increases.

Figure 27. Total time for S2CS dataset preprocessing using the Spark based implementation running over a Spark cluster with 10 nodes, using a varying number of cores per node



5.2. Topic Modeling

The objective of this section is to provide a measure of the performance of the different topic modeling techniques included in the TMT in terms of their training time and the quality of the topics obtained. The simulations that have been carried out constitute only a first approximation to the analysis of the available techniques and are subject to the following limitations.

- Regarding the quality of the models, the measure of **coherence** of their topics has been employed. In general, there is controversy about the usefulness of the coherence measures available in the literature, and there are even studies that show a negative correlation between some of these measures and the interpretability evaluated by human experts. For this reason, our experiments use two different coherence measures: `c_nmpi` and `u_mass`.
- Regarding the scalability of the topic modeling implementations, the different techniques available in the TMT require different architectures, so we have run our simulations on a server that has a graphic card for the training of the models based on `proLDA` and `CTM`¹³. Still, the training time required by `mallet` and neural techniques are not comparable. Lastly, the server does not have access to the Spark cluster, so it was not possible to include `spark LDA` in the comparisons.
- Note also that in this case, limitations of the memory of the GPU card available on the server impose a limit on the size of the training set, so we had to work with 1% of the documents available in the S2CS data set. More exhaustive experiments will be carried out after the TMT and IMT are deployed at the BSC infrastructure.

Despite these limitations, the experiments carried out allow us to draw a first series of conclusions, which will have to be subsequently validated by the users, both in terms of the

¹³ Intel(R) Xeon(R) Gold 5122 16 CPUs @ 3.60GHz 32GB and GPU Quadro P4000 8GB

quality of the topics obtained from the different techniques, and in terms of the usability of the tools derived from the training time they require (i.e., acceptability of the training time).

We start evaluating the results in terms of `c_npmi`, a coherence metric that relies on external sources. Figure 28 shows `c_npmi` coherence vs model training time for all simulated models using a diversity of training parameters, whereas Figure 29 includes just a subset of those models that have been selected according to their relevance to make a comparison. In all cases, the number of iterations allows for a good convergence of the algorithms.

Both figures show clear clusters in terms of performance and coherence, that can be each of them assigned to a different model type. Some conclusions can be extracted from an analysis of these results:

- With respect to variability of the models, we can see that, in spite of the non-deterministic nature of all topic modeling algorithms, the variability of the different runs is smaller within one model type than when compared to other models, which is why we were able to define the three clusters.
- In terms of coherence, Mallet is the algorithm showing the least dispersion followed by CTM and prodLDA. For the latter, more topics and more iterations seem to result in better coherence.
- With respect to training time, Mallet is more competitive than the neural models, and from the latter, ProDLDA is faster compared to CTM, mainly due to the addition of the time consuming construction process of the contextualized representations in CTM. It is noteworthy that this behavior in the neural models was to be expected when the training is carried out on a corpus with a large vocabulary size, as the S2CS dataset: at the end, both ProDLDA and CTM are not more than a neural model that reconstructs the input BoW; hence, the bigger the vocabulary, the more parameters we get, thereby the training becoming more difficult and prone to overfitting. The latter is especially emphasized with CombinedCTM, as the BoW reconstruction is added to the projection of the contextualized embeddings to the vocabulary space. In fact, the most competitive neural model w.r.t. coherence and topics' meaningfulness, that is the CTM model, requires over half an hour to train on this reasonably small dataset. Since training times scale linearly with the number of samples for these modes, this would imply over 50 hours of training time for the whole S2CS (for a fixed vocabulary size). In general, as we have already mentioned, more realistic simulations need to be carried out in the HPC infrastructure available at BSC. Note also that, in case Mallet remains competitive in terms of coherence, this is also a parallel implementation that can be sped up when running with more threads (up to 8 threads were used in the simulation).
- A major difference of neural implementations with respect to Mallet implementation is that (for a fixed infrastructure) both implementations scale roughly linearly with the number of training documents, but varying the number of topics has a close to linear impact on Mallet training time whereas the training time of neural implementations are barely affected. On the other hand, the number of parameters of the neural networks increase with the vocabulary size, implying also considerable increments on training

time. This implies that filtering the most relevant words in the dictionary is critical for these implementations.

Figure 28. Topic model comparison in terms of c_nmpi coherence for a variety of training parameters for Mallet, ProLDA, and CTM implementations

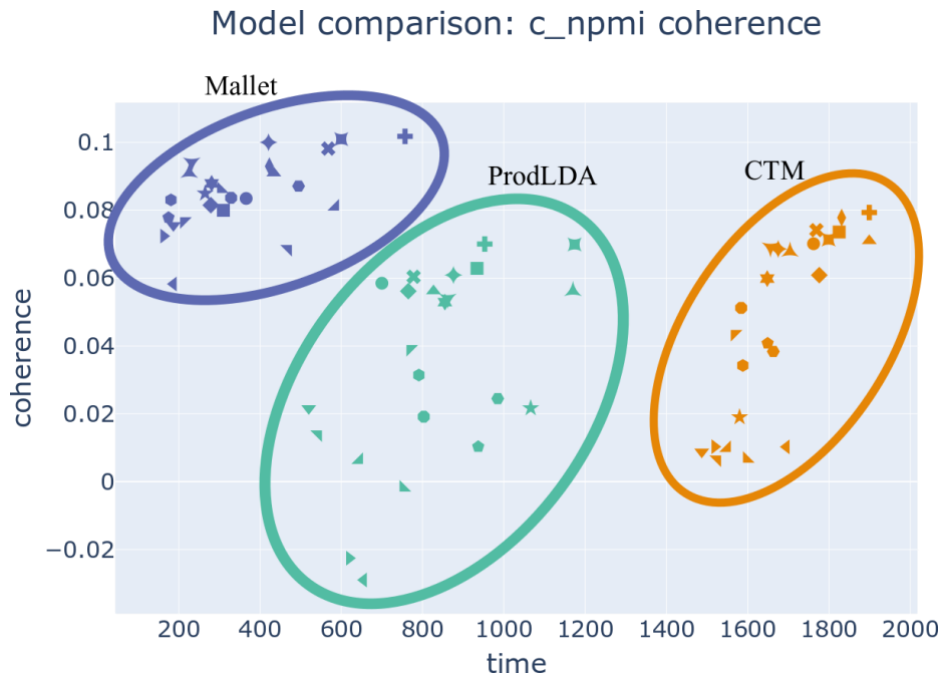


Figure 30 and Figure 31 display the same models mentioned previously, but showing model quality in terms of average u_mass coherence, a coherence measure that is based on the training set for calculation. The same cluster distributions can be observed, so most of the conclusions already described remain valid.

However, an important difference is that u_mass coherences have an opposite interpretation: the lower the u_mass values, the better the topics. This implies that according to this metric Mallet would show the worst results of all methods. This fact, together with the already discussed controversy about the use of coherence metrics for topic model evaluation, reinforces the need to carry out subjective experiments within the IntelComp project to assess the quality of the topic models obtained from all these techniques. In any case, note that all techniques will be available for use in the IMT, so in the end it will be up to the Intelcomp user/operator to explore the quality of the topics for each specific use case and training data set.

Figure 29. Topic model comparison in terms of `c_npmi` coherence for a variety of training parameters for Mallet, ProDLDA, and CTM implementations. A selection of the most relevant models for comparison is included (with the legend showing the settings for each experiment)

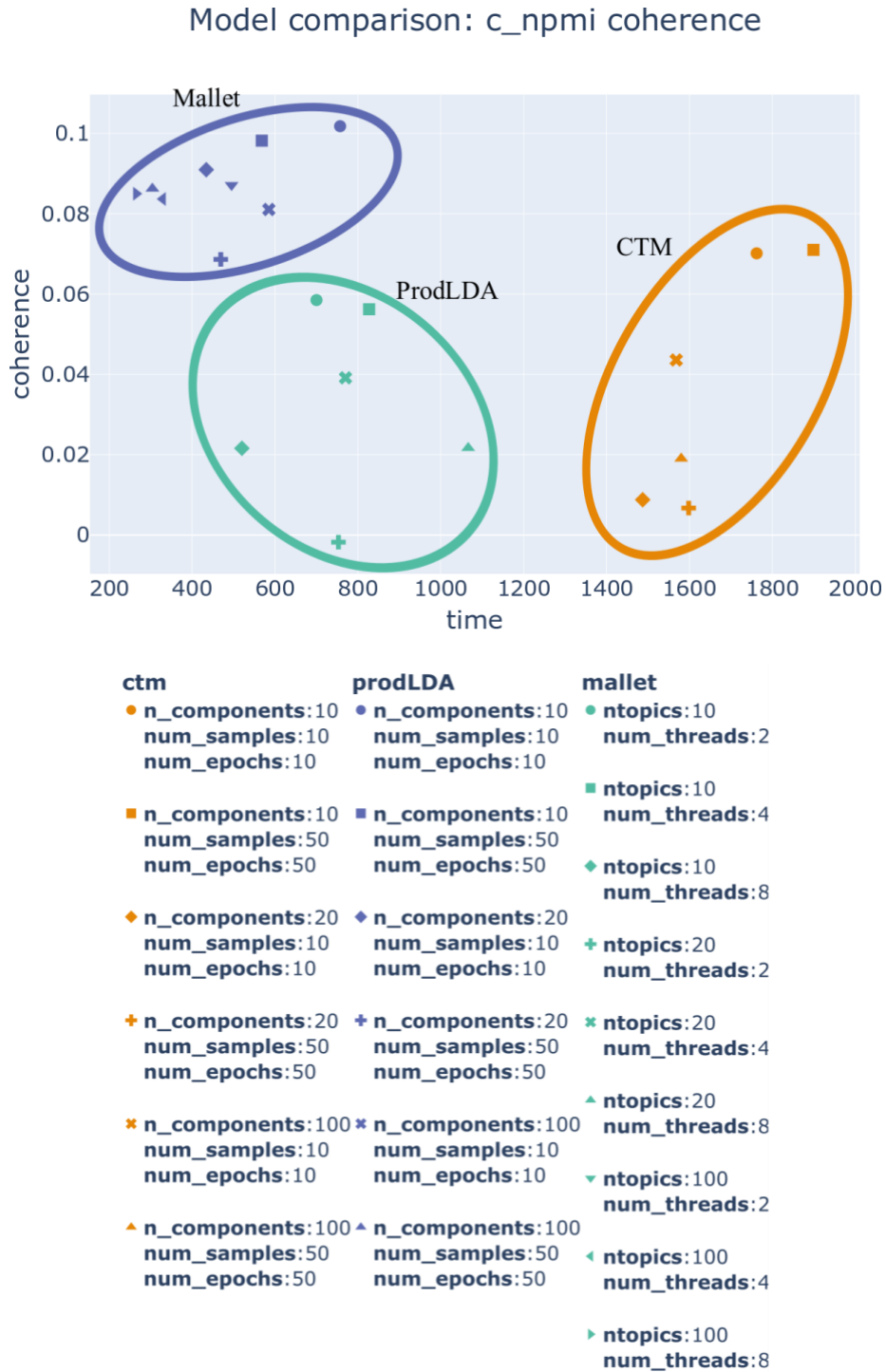
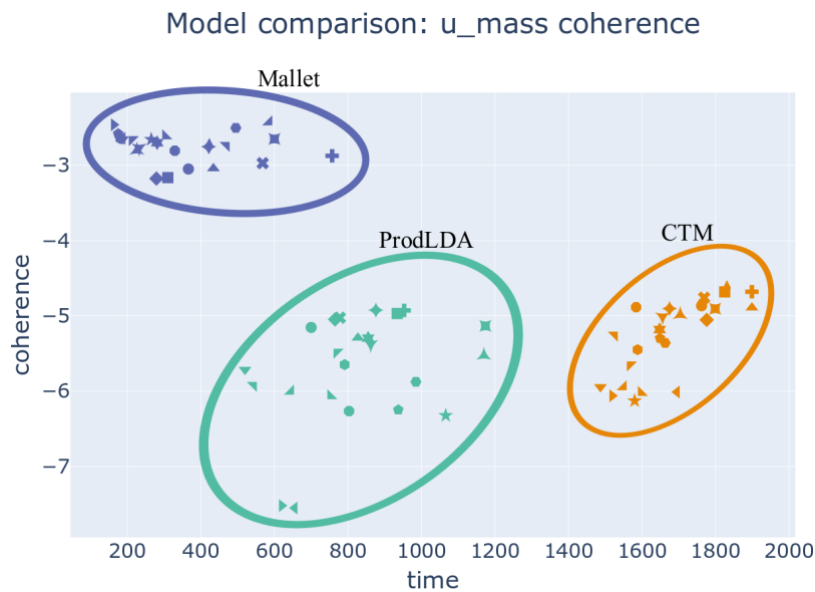
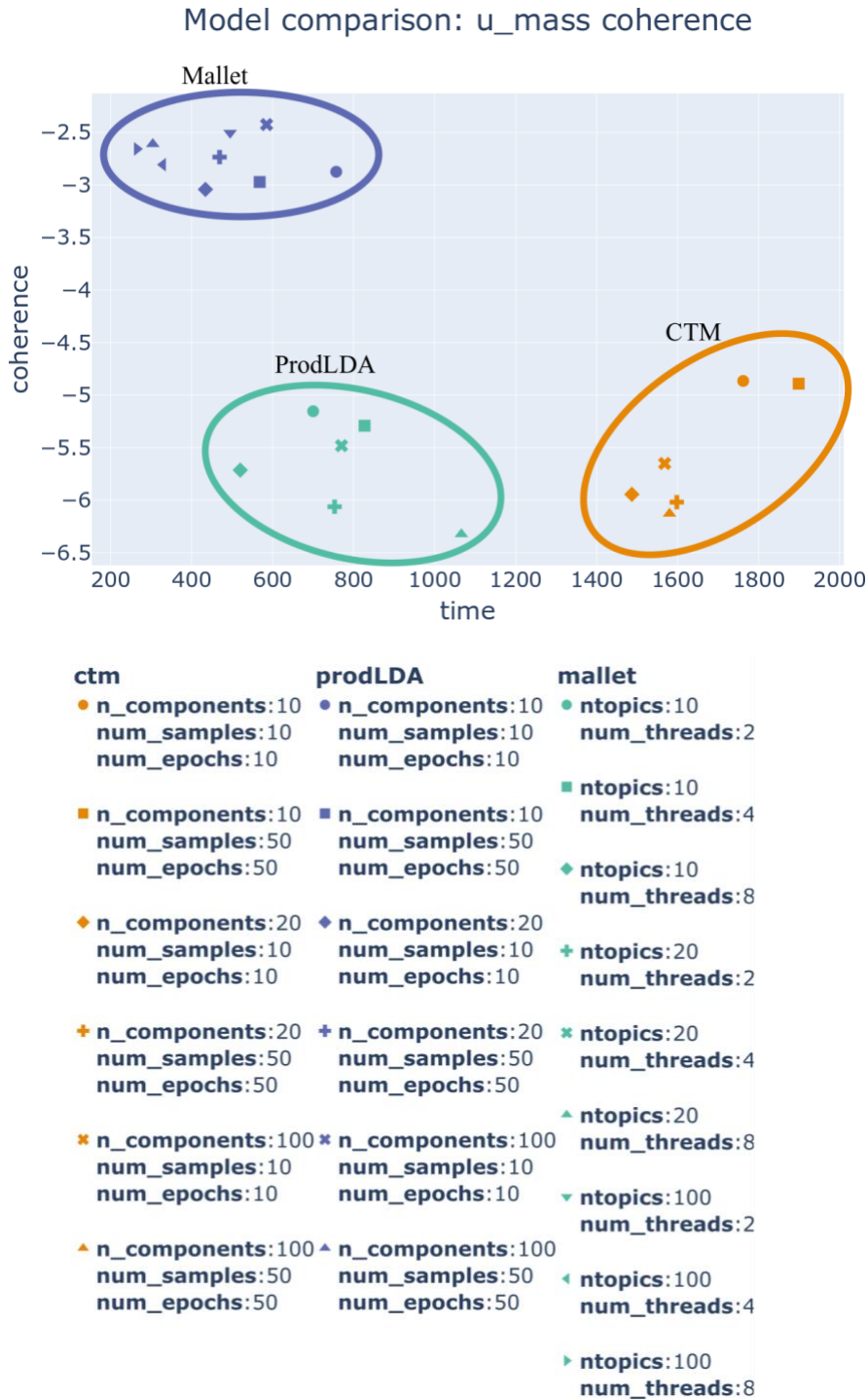


Figure 30. Topic model comparison in terms of u_{mass} coherence for a variety of training parameters for Mallet, ProDLDA, and CTM implementations



It is not clear which solution offers better performance given the incompatibilities of the results of the two coherence measures. However, this is not a surprising finding in the context of topic modeling, as it is known that different coherence metrics can lead to contradictory results, and that in some cases coherence values can even show a bad correlation with human interpretability [23], [27]. For these reasons, human evaluation in the context of IntelComp will be a very valuable resource to assess the quality of topic models.

Figure 31. Topic model comparison in terms of u_{mass} coherence for a variety of training parameters for Mallet, ProDLDA, and CTM implementations. A selection of the most relevant models for comparison is included, with a legend showing the settings for each experiment



6. CONCLUSIONS AND FUTURE WORK

Topic modeling is a core technique of the IntelComp project to enrich the information from the different sources of information to be analyzed, and to be able to automatically identify the predominant themes in each data set and/or comparing them between different data sets. To do this, the objective of Task 3.5 consisted of generating a series of tools that allow the construction of topic models and their adjustment following a user-in-the-loop approach.

This document describes the functionality of the toolbox developed within Task 3.5 of IntelComp: the Topic Modeling Toolbox (TMT):

- Description of the document preprocessing processes for the generation of training datasets. For greater efficiency, preprocessing can be parallelized using a Spark cluster or a powerful multicore machine using the Python Dask library. Both implementations have been compared in terms of efficiency, arriving at the conclusion that Spark implementation can lead to shorter processing times, if running on a large cluster. For moderate-sized datasets, Dask implementation running on a Multicore server with 20-40 CPUs presents also acceptable running times.
- Description of the topic modeling techniques included in the TMT: Mallet, SparkLDA, proLDA, CTM, and two-level hierarchical models based on any of the above. The working principles of these implementations have been presented and compared in terms of training times and coherence. However, these results are somewhat limited and need to be further validated in the project with respect to 1) actual running times in the IntelComp production environment, and 2) quality of the topic models (calculated average coherences are not conclusive in this respect).
- Description of a battery of tools for the visualization, evaluation and curation of models by experts: e.g., selection of new stopwords, fusion or fission of topics, elimination of noisy topics, etc. For this, an abstraction of topic models has been defined that allows working in a unified way with the models obtained with any of the available training techniques. The usability of these tools needs to be assessed by the final users in the context of the living labs. Further developments to adjust them, or to create new ones, are expected to take place during the next 3-6 months and will be addressed using an agile approach as indicated in the IntelComp methodology guidelines.

In addition to this, the TMT software has been documented providing the necessary information for integration with other IntelComp components:

- All services have been encapsulated in dockers to facilitate their use by other IntelComp components. Instructions for the use of these dockers have been included with some examples to facilitate the use of TMT as part of the IMT backend service.
- Two front-ends have been developed for the use of the tool in a 100% python environment. One of these UIs is based on interaction with the user through the command line, while the second consists of a graphical interface based on PyQt6. These UIs serve as demonstration tools to define the workflows that need to be integrated in

the IMT, and illustrate the API commands that need to be executed to obtain specific services.

Although there are still some aspects of efficiency improvement in the development of the TMT itself (current open issues in the Github project), the implementation of the TMT can be considered essentially complete. In the future, the main activity will be its deployment within the BSC infrastructure and integration with other IntelComp tools. In this sense, the integration with the IMT is already advanced, having benefited from the design of the TMT that decouples the services from the user UI.

On the other hand, it is to be expected that the use of the tools developed by the end users will allow the identification of needs for change and improvement that imply changes in the developed SW and/or the inclusion of new functionalities. As already mentioned, an agile methodology will be used for this, according to the methodology defined by IntelComp, carrying out sprints of 2-3 week duration aligned with the work of the Living Labs (IntelComp's WP6). Changes in the TMT will be continuously published in the corresponding open Github project. From the point of view of the IMT, improvement of services will be transparent to the tool; however, if new functionalities and/or workflows are requested, these will in general affect both the back-end services and front-end visualization; consequently, there will be a need to address these requests by both the TMT and IMT development teams.

REFERENCES

- [1] M. Vázquez, J. Pereira-Delgado, J. Cid-Sueiro and J. Arenas-García, "Validation of scientific topic models using graph analysis and corpus metadata," *Scientometrics*, 2022.
- [2] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, no. 3, pp. 993-1022, 2003.
- [3] Baruffaldi, S., et al., "Identifying and measuring developments in artificial intelligence," *OECD Science, Technology and Industry Working Papers*, no. 2020/05, 2020.
- [4] A. Bonaccorsi, M. N. and F. A. Massucci, "Exploring the antecedents of interdisciplinarity at the European Research Council: a topic modeling approach," *Scientometrics*, 2022.
- [5] Talley E. M., et al., "Database of NIH grants using machine-learned categories and graphical clustering," *Nature methods*, pp. 443-444, 2011.
- [6] L. G. Nichols, "A topic model approach to measuring interdisciplinarity at the National Science Foundation," *Scientometrics*, pp. 741-754, 2014.
- [7] Yamashita, I., et al., "Measuring the AI content of government-funded R&D projects: A proof of concept for the OECD Fundstat initiative," *OECD Science, Technology and Industry Working Papers*, no. 2021/09, 2021.
- [8] D. M. Blei and J. D. Lafferty, "Dynamic Topic Models," in *Proc. Intl. Conf. Machine Learning*, 2006.
- [9] T. L. Griffiths, M. I. Jordan, J. B. Tenenbaum and D. M. Blei, "Hierarchical topic models and the nested Chinese restaurant process," in *Advances in Neural Information Processing Systems*, 2004.
- [10] D. M. Blei and J. D. Lafferty, "A correlated topic model of science," *The annals of applied statistics*, vol. 1, no. 1, pp. 17-35, 2007.
- [11] A. Srivastava and C. Sutton, "Autoencoding variational inference for topic models," arXiv preprint arXiv:1703.01488, 2017.
- [12] F. T. S. Bianchi and D. Hovy, "Pre-training is a hot topic: Contextualized document embeddings improve topic coherence," in *Proc. 59th Annual meeting of the ACL and 11th Intl. Joint Conf. Natural Language Processing*.
- [13] F. Bianchi, S. Terragni, D. Hovy, D. Nozza and E. Fersini, "Cross-lingual contextualized topic models with zero-shot learning," in *Proc. 16th Conf. European Chapter of the ACL*, 2021.

- [14] M. Grootendorst, "BERTopic: Neural topic modeling with a class-based TF-IDF procedure," arXiv preprint arXiv:2203.05794, 2022.
- [15] M. Steyvers and T. Griffith, "Probabilistic Topic Models," in *Handbook of latent semantic Analysis*, Mahwah, NJ, 2007.
- [16] Chaudhary, Y., et al., "TopicBERT for energy efficient document classification," arXiv preprint arXiv: 2010.16407.
- [17] Y. Miao, L. Yu and P. Blunsom, "Neural variational inference for text processing," in *Proc. Intl. Conf. Machine Learning*, 2016.
- [18] R. Ranganath, S. Gerrish and D. Blei, "Black box variational inference," in *Proc. 17th Intl Conf. on Artificial Intelligence and Statistics*, 2014.
- [19] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, pp. 1771-1800, 2002.
- [20] D. Card, C. Tan and N. A. Smith, "Neural models for documents and metadata," in *Proc. 56th Annual Meeting of the ACL*, Melbourne, Australia.
- [21] Higgins, I., et al., "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," in *Proc. ICLR*, 2016.
- [22] H. M. Wallach, I. Murray, R. Salakhutdinov and D. Mimno, "Evaluation methods for topic models," in *Proc. 26th Intl. Conf. Machine Learning*, 2009.
- [23] J. Chang, S. Gerrish, C. Wang, J. Boyd-Graber and D. Blei, "Reading tea leaves: How humans interpret topic models," in *Advances in Neural Information Processing Systems*, 2009.
- [24] D. Newman, J. H. Lau, K. Grieser and T. Baldwin, "Automatic evaluation of topic coherence," in *Proc. 2010 Annual Conf of the North American Chapter of the ACL*, 2010.
- [25] D. Mimno, H. Wallach, E. Talley, M. Leenders and A. McCallum, "Optimizing semantic coherence in topic models," in *Proc. 2011 Conf. on Empirical Methods in Natural Language Processing*, 2011.
- [26] N. Aletras and M. Stevenson, "Evaluating topic coherence using distributed semantics," in *Proc. 10th Intl. Conf. Computational Semantics*, 2013.
- [27] M. Röder, A. Both and A. Hinneburg, "Exploring the space topic coherence measures," in *Proc. 8th ACM Intl. Conf. on Web search and Data Mining*, 2015.
- [28] R. Ding, R. Nallapati and B. Xiang, "Coherence-aware neural topic modeling," in *Proc. 2018 Conf. Empirical Methods in Natural Language Processing*, Brussels, 2018.

- [29] W. Webber, A. Moffat and J. Zobel, "A similarity measure for indefinite rankings," *ACM Trans. on Information Systems*, vol. 28, no. 4, pp. 1-38, 2010.

- [30] S. Terragni, E. Fersini and E. Messina, "Word embedding-based topic similarity measures," in *Proc. 26th Intl. Conf. Applications of Natural Language to Information Systems*, 2021.

- [31] D. M. Blei, "Probabilistic Topic Models," *Communications of the ACM*, vol. 55, pp. 77-84, 2012.