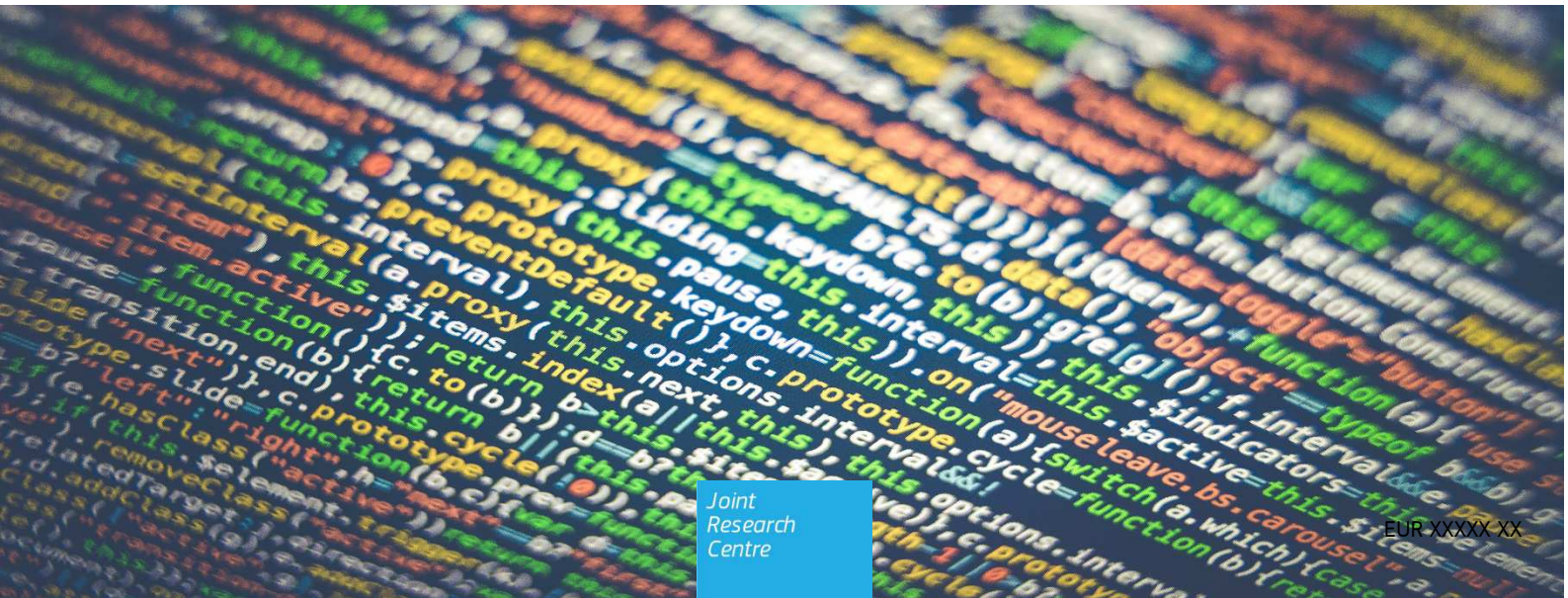
European Commission

Existing Meta Data Standards to be adopted for open information processing applications

Report for ISA² Action 2018.02 – Processing Open Source Data with Exchangeable Components

2019, V0.5



Joint Research Centre

EUR XXXXX XX

This is a project related publication by the ISA^2 Action 2018.02 – Processing Open Source Data with Exchangeable Components. Unit i.03 of the Joint Research Centre (JRC) manages the action. The JRC is the European Commission's science and knowledge service. It aims to provide evidence-based scientific support to the European policymaking process. The document's content does not imply a policy position of the European Commission. Neither the European Commission nor any person acting on behalf of the Commission is responsible for the use that might be made of this publication.

Contact information
Address: JRC I.3, TP440, Via Enrico Fermi 2749, 21027 Ispra (VA), Italy
Email: bertrand.de-longueville@ec.europa.eu

ISA^2 Home Page
https://ec.europa.eu/isa2/home_en

Action 2018.02 Home Page
https://ec.europa.eu/isa2/actions/processing-open-source-data-exchangeable-components_en

Action 2018.02 EC Joinup
https://joinup.ec.europa.eu/collection/isa-action-201802-processing-open-source-data-exchangeable-components

Contents

1 Introduction

In the recent two decades, one could observe an emergence of various integrated frameworks and toolkits designed for assembling natural language text processing chains. For the purpose of this report, we decided to select a subset of such frameworks in order to examine the meta-data standards for processing open source information. We used the following criteria in the selection process:

Coverage: The range of the processing components covered

Popularity: The number of users, initiatives and projects which use the framework for research or operationally

Project Health: The frequency of how new upgrades and features are released.

Multilinguality: The range of languages and language-specific resources covered

Portability: How usable the software is in different environments.

Modularity and Workflow Management: How modular the software is and how the software supports assembling dedicated text processing chains. Especially important is the combination of core processing components with clearly defined I/O specifications.

In principle, two frameworks are specialised for natural language processing and stand out vis-a-vis the criteria above, namely Apache UIMA (Unstructured Information Management Architecture) and GATE (General Architecture for Text Engineering). We will briefly describe both frameworks in the following chapters. Additionally, we will describe the KNIME Analytics Platform. This platform provides a framework aimed more broadly at data processing than only natural language processing.

2 Apache UIMA – Unstructured Information Management Architecture

Apache UIMA [1] is an implementation of the UIMA standard for content analytics, developed originally by IBM, which provides a component software architecture for the development, discovery, composition, and deployment of multi-modal analytics for the analysis of unstructured data and integration thereof with search technologies.

UIMA has been frequently used to build text mining applications, e.g., where one ingests plain text collections as input and identifies entities, such as persons, places, organizations; or relations, such as works-for or located-at in those texts. UIMA enables applications to be decomposed into clear-cut components and assembling processing chains, e.g., "language identification", followed by ”language specific segmentation", followed by "sentence splitting", and finally followed by "entity mention detection”. Each component implements interfaces defined by the UIMA framework and provides self-describing metadata via XML descriptor files. The framework facilitates the management of these components and the data flow between them. Native components are written in Java or C++ and the data that flows between components is designed to support efficient mapping between these languages.

UIMA also provides capabilities to wrap components as network services and facilitates scaling through replicating processing pipelines over a cluster of networked nodes. Apache UIMA includes APIs and tools for creating analysis components, including Text Mining-relevant ones, e.g., tokenizers, text summarizers, text categorizers, parsers, named-entity detectors etc. Many Apache UIMA community-created components exist. Apache UIMA also supports the development and integration of analysis components developed in different programming languages (e.g. Perl, Python, and TCL). It is worth mentioning that Apache UIMA supports the development of multi-modal analytics, which apart from text, includes processing of audio and video material. Apache UIMA is highly regarded for its modularity, scalability and support for developing components in different languages, including Java, C++ and Python.

3 GATE – General Architecture for Text Engineering

GATE [2] is a general-purpose infrastructure for developing and deploying software components for processing human language and provides a workbench for Natural Language Processing (NLP) (in a similar fashion Eclipse is a workbench for software developers).

It is circa 15 years old and evolved over time and has a large community of active users. It is used for implementing all types of text processing applications. In GATE philosophy, the elements of software systems that process natural language are broken into various types of components, known as resources.

Components are reusable software chunks with well-defined interfaces and are a popular architectural style to modularize a software system. GATE components are specialised types of modules and come in three flavours:

a) Language Resources (LRs), which represent entities such as lexicons, corpora or ontologies

b) Processing Resources (PRs), which represent entities that are primarily algorithmic, such as parsers, generators or n-gram modellers

c) Visual Resources (VRs), which encompass components for visualisation and editing being part of GUIs.

The strength of GATE comes due to the wide range of language processing components that come bundled with the framework, in particular for developing information extraction applications and covering resources for processing texts in many different languages. Yet, the essential part of GATE is *GATE Developer*, a development environment that provides a rich set of graphical interactive tools for the creation of processing pipelines for processing human natural language and carrying out measurement and maintenance thereof. These processing pipelines can be exported to other GATE sub-frameworks for distributed quality control and parallelisation, etc.

Another GATE sub-framework, namely, *GATE Embedded*, is an object library optimised for inclusion in diverse applications giving access to all the services used by GATE Developer.

4 KNIME Analytics Platform

The KNIME Analytics Platform is an open source software that allows creating data workflows (data pipelines) [3]. Besides the KNIME Analytics Platform, which is available as open source, there is a commercial server product, called KNIME Server which can be used to deploy analytical applications for operational use.

The software contains a visual editor that allows the user to assemble processing steps (called Nodes) into a data workflow. Currently, there are over 2000 different processing nodes provided to the user to be used for assembling a data workflow. The workflow concept follows the classical ETL (Extract Transform Load) approach where data from different source is gathered, transformed into a unified form and then loaded into an analytical tool for further processing and analysis. KNIME is especially popular in the life science domain where data science is heavily used for research purposes.

According to the selection criteria we outlined in the introduction chapter of this report, KNIME covers a very wide range of processing needs. The software is very popular, and the user base of the software is growing. Members of our Action's stakeholder group have reported that users of public authorities are also using the software. The software scores also strong in other criteria, such a Project Health, Portability, and Modularity and Workflow Management. The processing of data in the field of natural language processing however is not per-se a core domain of KNIME. The KNIME documentation lists a Text Processing section [4] which contains a technical report [5] about the text processing capabilities. As can be seen from the examples in the report, KNIME text processing is driven by life science use cases where pharmaceutical companies process patent applications and others documents mostly in the domain of Competitive Intelligence.

KNIME is a promising platform in the field of open source data processing. However, to make it applicable for processing open source data with means of natural language processing more capable processing nodes need to be added to the platform.

5 Other Toolkits and Frameworks

Apache UIMA and GATE are the frameworks of primary importance for our report. However, there are other frameworks and toolkits worth mentioning in this context. The reason is that components provided by these frameworks might be reused for natural language processing or higher-level text mining task and potentially integrated in Apache UIMA or GATE. In particular, we list some of these tools here since the respective performance of some of their core modules in terms of accuracy and/or coverage might be superior vis-a-vis the ones available in GATE/UIMA or elsewhere. Some of them provide a wide range of core components with different implementations, whereas others are model-based, i.e., models underlying specific components are trained using some specific machine-learning paradigm.

5.1 OpenNLP

The Apache OpenNLP library [3, pp. 237-269] supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, language detection, coreference resolution and document categorization. These tasks are usually required to build more advanced text processing services. This library is based on machine learning techniques; in particular, it includes a large number of pre-built maximum entropy and perceptron-based machine learning models for a variety of tasks and languages, as well as the annotated text resources that those models are derived from. OpenNLP provides also functionality to train new models and evaluate them. All components and functionalities are accessible through APIs or can be run from the command line.

5.2 NLTK

NLTK- the Natural Language Toolkit [4] is a collection of independent core modules with clear-cut I/O specifications and data sets supporting research and development in the area of Natural Language Processing. NLTK toolkit is written in Python and is open source. NLTK supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities. Each core module defines relevant basic data types and processing tasks that are used within the toolkit. For instance, the tokenizer module provides basic classes for processing individual elements of text, such as words or sentences. The tree module defines data structures for representing tree structures over text, such as syntax trees and morphological trees. Furthermore, the probability module implements classes that encode frequency distributions and probability distributions, including a variety of statistical smoothing techniques. NLTK is primarily intended to support research and teaching in NLP and closely related areas.

5.3 Stanford NLP and Stanford CoreNLP

Stanford NLP and Stanford CoreNLP [5] are libraries core text processing modules that cover most common natural language analysis steps and which are written in Python and Java. Stanford tools can segment a text into sentences and words, lemmatise the latter ones, compute the respective part-of-speech and morphological features and syntactic structure of entire sentences, etc. Noteworthy, Stanford NLP provides neural models supporting over fifty languages (with interfaces to train models for new languages available). Stanford NLP/CoreNLP uses a uniform interface for adding annotations for a

given text by each core component and annotations computed by third-party components can also be easily integrated, which made the Stanford toolkits popular in the community in the context of assembling NLP annotation pipelines.

5.4 DKPro

DKPro is a community of projects focussing on building re-usable Natural Language Processing software. In particular, DKPro Core [6] addresses tasks that are commonly referred to as linguistic pre-processing, e.g. part-of-speech tagging, lemmatisation, parsing, etc. Within DKPro Core, a steadily growing set of third-party tools and some original tools for such tasks have been wrapped into interoperable and interchangeable components for the Apache UIMA framework so they can be used interchangeably in UIMA processing pipelines.

Interestingly, many of the core modules from Stanford NLP, OpenNLP, GATE and LingPipe were integrated. DKPro Core builds heavily on *uimaFIT* (a library of Apache UIMA), which allows for rapid and easy development of NLP processing pipelines, for wrapping existing tools and for creating original UIMA components. Other projects, e.g. DKPro WSDm, DKPro Similarity, DKPro WSD focus on the development of tools for word sense disambiguation, text similarity algorithms and text classification respectively.

5.5 Mallet

MALLET [7] is an open source Java library for statistical natural language processing, which provides tools for carrying out higher-level natural language processing tasks, including:

— document classification,

— clustering,

— topic modelling,

— information extraction, etc.

It also provides routines for converting text to features and numerical representations, a wide variety of machine learning algorithms, tools for sequence tagging for applications, etc. MALLETs main strength lies in efficient implementation of the various tasks and routines.

5.6 NLP4J

Natural Language Processing for JVM languages (NLP4J) [8] provides NLP components for carrying out the following core linguistic analysis tasks:

— Tokenization

— Morphological analysis

— Part-of-speech tagging

— Named-entity recognition

— Syntactic parsing.

Some of the components are based on machine learning techniques and NLP4J provides functionalities to train the respective models on data provided. by the user. NLP4J comes equipped with models for processing English texts

5.7 LingPipe

LingPipe is a suite of Java libraries for the linguistic analysis of texts [9], with a particular focus on information extraction, i.e., named-entity recognition, entity linking, relation extraction, and other standard NLP tasks such as language detection, part-of-speech tagging, syntactic parsing, text classification, clustering and sentiment analysis. Although it is no longer under active development, it is listed here since many wrappers for LingPipe component exist, and are still in use.

5.8 spaCy

SpaCy is an open-source library for carrying out various core NLP task written in Python [10]. The pool of components provided encompasses:

— tokenization

— part-of-speech tagging

— syntactic parsing

— lemmatization

— sentence boundary detection

— named-entity recognition

— text similarity computation

— text classification

— pattern matching based on linguistic information

— others

While spaCy supports processing texts in up to 50 languages, it is not designed to allow exchanging components with external frameworks. Furthermore, it is less configurable than Stanford NLP, Open NLP and NLTK since it is more industry-oriented vis-a-vis the more research-oriented toolkits like the ones mentioned before. Noteworthy, spaCy comes with pre-trained statistical models and word vectors (word embeddings) for multiple languages.

5.9 AllenNLP

AllenNLP [11] is an Apache 2.0 NLP research library, built on PyTorch [12], for developing deep learning models for a wide variety of higher-level text mining tasks. In particular, AllenNLP include reference implementations of models for Semantic Role Labelling and Question Answering.

5.10 NLP Architect

Analogously to the AllenNLP, NLP Architect [13] is an open-source Python library for exploring deep learning topologies and techniques for natural language processing and natural language understanding tasks. It builds on top of TensorFlow and comes with pre-trained models for named-entity recognition, parsing, intent extraction and question answering.

5.11 FLAIR

Flair [14] is another Python-based NLP toolkit that covers functionalities such as named entity recognition, part-of-speech tagging, sense disambiguation and text classification. It exploits PyTorch for the purpose of learning models for the various NLP tasks. It comes with a library of word embeddings, also multilingual ones, i.e. one model covers many languages.

6 Essential elements from GATE and Apache UIMA to derive ISA^2 standard for processing open source information

In this section, we describe the relevant elements of GATE and UIMA for the elaboration of meta-data standards for processing open source information.

Both GATE and UIMA provide data and processing models, respective APIs, and a GUI-based tool that facilitate assembling pipelines of arbitrary NLP components. Either framework provides a certain level of genericity, i.e., means to easily plug-in new and external components and exploit and merge their results with other components. Furthermore, there is a fair amount of interoperability between the two frameworks through provision of wrappers and interfaces enabling UIMA run GATE components and vice-versa.

Both GATE and UIMA facilitate assembling text document processing pipeline architectures. UIMA is a more generic framework providing containers with standard I/O interfaces to plug in "engines". The processing is not only limited to analysing texts but encompasses any unstructured information, and where the focus is on performance, scalability (native support for distributed processing via web services) and strong support to integrate components written in different programming languages. UIMA is certainly the most evolved and comprehensive architecture regarding the infrastructural capabilities. UIMA can be seen as a generalisation of GATE. GATE, on the other hand, is considered a framework for assembling specifically text processing engines that comes equipped with a large reservoir of pre-cooked solutions and linguistic/processing resources, and is relatively easy to configure due to its less generic character.

6.1 Annotation Models

6.1.1 GATE

GATE uses the notion of feature maps, which are used for annotation purposes. Features in GATE are simply attribute-value pairs, where attributes are strings, and values are arbitrary Java objects. Feature maps are used to associate data with individual annotations, documents and document corpora.

Document annotation is considered in GATE as forming a directed acyclic graph. The nodes of the graph are text offsets into the document (between characters) and the edges are annotations of all sorts between nodes. Each annotation consists of five elements:

- `annoId`: an (int) identifier
- `annoType`: a type (represented as string), eg. token, sentence, etc.
- `annoStart`: start Node
- `annoEnd`: end Node
- `annoFeatures`: a feature map adding additional information

In GATE, the feature map above may include any features, with any values. A text in GATE can be associated with annotations and metadata, i.e., a Document consists of:

- `docContent`: text of the document
- `docFeatures`: meta-data attributes (a feature map)
- `docAnnotations`: a non-empty set of annotations (**set of annotation sets**)

Analogously, a Corpus consist of:

- `corpusDocs`: a non-empty set of documents
- `corpusFeatures`: accompanying metadata (feature map), e.g., information on who created the corpus, and/or what tool was used

Noteworthy, annotations are grouped into multiple annotation sets. This feature might be used to associate each annotation set with a different meta-data, e.g., on who and/or what tools were used to generate them.

6.1.2 UIMA

The first important fact is that UIMA can produces annotations for different types of content, e.g., text, audio, video, etc. For annotations so-called "type hierarchies" are used, i.e., own type classes and subclasses are defined. In contrast to GATE's approach, feature structures used for creating annotations are strictly typed and different types of documents have different kinds of annotations specified using a concrete type class. Each type class has a set of features and new classes can be derived using single class inheritance. Features on the other hand have a type class, and cardinality (lower and upper bound). Features come in two varieties, namely, "attributes", whose type must be one of the primitives classes, and "references", whose type is a class. UIMA provides a base type system from which other type systems can be derived and which is assumed to be common across all UIMA-based applications. In particular, the base type system defines primitive types like strings, integers, floats, etc.

The base type system in UIMA defines the notion of an Annotation, which in principle is:

- an object of certain type
- with regional references, e.g., offsets for textual data
- on a Subject of Analysis, so called "SOFA" (e.g., text)

Similar to GATE, the annotation is standoff on text documents using some positional information, however, UIMA adds an additional layer of abstraction, namely, subjects of analysis and regional references.

The annotations are represented with "Annotation" classes in UIMA, whereas each of the annotations uses "`SofaReference`" pointers to refer to the content being annotated so that references can be shared by annotations.

While in GATE, one "annotates" documents and document corpora, in UIMA a more generic concept of a Common Analysis Structure (CAS) is used, namely, a sort of self-contained bundle consisting of:

- an artifact representing any kind of unstructured data,
- related annotations and metadata (in UIMA annotations are part of the metadata)
- the type system used by the annotations

It is important to emphasize that artifacts may not necessarily be single documents, but they could potentially be a set of multiple objects of different modalities. Annotations are grouped into a set of views on their artifacts, which provide a particular interpretation on the given artifact. The introduced CASs constitute I/O structures that UIMA components consume and produce.

6.2 Component Model

This section provides some details on the component models used in GATE and UIMA.

6.2.1 GATE

GATE components are subdivided into three types:

- `LanguageResources` (LRs) which encompass entities such as lexicons, corpora or ontologies,
- `ProcessingResources` (PRs) which encompass entities that are primarily algorithmic, such as parsers, generators or ngram modellers, and
- `VisualResources` (VRs) which represent visualisation and editing components that participate in GUIs.

A component (resource) in GATE may be implemented using a variety of programming languages and/or databases, but they must be represented to the framework as a Java class. The entire set of components integrated in GATE is referred to as CREOLE: a **C**ollection of **RE**usable **O**bjects for **L**anguage **E**ngineering.

Component implementations are grouped together as 'plugins' and stored typically in a single JAR file or in binary serialised form (linguistic resources) in the local file system. Once a plugin is loaded into GATE, it looks for an XML configuration file in order to identify which resources are declared and which classes need to run. Then it initializes the respective resources, which are then added to the CREOLE register. This register is used by GATE to return Instantiations of the registered resources on request of the user.

The GATE framework can be seen as a backplane into which users plug CREOLE components, which are loaded on starting the system. In principle, the backplane carries out the following four tasks:

- component discovery, (loading and reloading,
- management and visualisation of native data structures for common information types,
- data storage
- process execution

The framework together with a set of components constitutes a deployment unit, which can be embedded in other applications.

All GATE resources are Java Beans.

Processing resources can be combined into applications, which are called Controllers in GATE. There are two "pipeline" control flows in GATE:

- simple pipelines that group a set of processing resources together in order and execute them in turn, and
- corpus pipelines that are specifically applied to corpora, where each document is processed by all processing resources in turn

Controllers provide ways to define conditions that need to be matched by documents in order to be processed. Controllers may be equipped with methods that are called at the start and end of the execution of all processing resources in the pipeline. Other parameters of the controllers allow specifying a timeout parameter to determine the maximum amount of time allowed for the processing of a document.

Language Resources can be stored in data stores, of which two types are available:

- Serial data stores based on Java's serialisation system, and

6.2.2 Lucene data stores which utilise the annotation indexing and retrieval capabilities of Lucene [15] UIMA

The atomic processing unit in UIMA is called "Annotator". Annotators analyse artifacts and create additional data (metadata) about that artifact (e.g., names found in a textual document). They do not need to be aware about the details of their deployment and interaction with other annotators, etc.

An Analysis Engine (AE) is a program that analyses artifacts (e.g. documents) and infers information from them. They are constructed from annotators. An AE may contain just a single annotator, which is referred to as a Primitive AE, or it may be a certain composition of multiple annotators, which is referred to as an Aggregate AE. At the programming level, there is no difference between primitive and aggregate AEs. Both implement the same programming interface and can be used interchangeably.

6.3 Adapting features of KNIME, GATE, UIMA and other toolkits

Within the context of the ISA^2 action on developing a good practice set of meta-data standards for open source information processing applications we can conclude:

- A core requirement is that analysts can quickly test a processing pipeline against open source data. For this requirement, the KNIME Analytics Platform represents an excellent solution, since it provides a visual editor which allows rapid prototyping. In addition, the modular approach allows to plug in additional functionality to process specific data. The built-in modules for natural language processing seem to be rather basic. Therefore, an upgrade with better language processing capabilities is required.
- All clear-cut I/O interfaces defined for core linguistic processing tasks need to correspond to core functionalities of interest to end-users. End-users are users who design higher-level applications for processing open source information and are interested in exploiting and analysing the data. To be more precise, it is out of scope to define abstractions for core linguistic analysis such as tokenization, part-of-speech tagging, and syntactic parsing. In contrast, end-users are interested in core functionalities that are needed to design interoperable interfaces, such as dictionary look-up, name recognition and event extraction.
- While GATE provides concepts and processing modules to work with text in order to create NLP solutions it is, however, mostly restricted to this text-only domain.

Compared to KNIME it does provide less tools to extract text from binary formats or from databases. GATE is at its core a workbench to deal with NLP problems. GATE's practical appeal for rapid OSINT prototyping that needs an end-to-end solution is limited where input texts can exist in many different representations.

- The plugin-model of GATE to add functionality to the platform is compared to KNIME more specific to NLP functions and therefore does not have the same wide appeal to process data from different sources.
- UIMA provides a level of genericity that requires a substantial overhead to make external processing resources compliant with it. The strict typisation mechanism used by UIMA represents a clear benefit. However, the typisation should be strictly limited to the particular domain of processing open source information and with some types declared invariant.
- Going forward, it is needed to define a pragmatic abstract layer for natural language processing which allows to produce product agnostic processing modules. In addition, an adaptation layer to integrate these modules into KNIME would lead to synergies between more capable text processing and the rapid prototyping and visual editor capabilities offered by KNIME. The rapid prototyping characteristic of KNIME is supported by a wide range of processing nodes which allow to extract and transform source documents to texts and also provide powerful analytical functions to make sense of the processed data.

Bibliography

[1] R. &. G. I. Eckart de Castilho, "A broad-coverage collection of portable NLP components for building shareable analysis pipelines," in *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, 2014.

[2] H. Cunningham, V. Tablan, A. Roberts and K. Bontcheva, "Getting More out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics," *PLoS computational biology,* vol. 9, no. 2, p. e1002854, 2013/02/01.

[3] K. AG, "KNIME," [Online]. Available: https://knime.com.

[4] K. AG, "KNIME Text Processing," [Online]. Available: https://www.knime.com/knime-text-processing.

[5] K. Thiel and M. Berthold, "The KNIME Text Processing Feature - An Introduction," 2012. [Online]. Available: https://www.knime.com/sites/default/files/knime_text_processing_introduction_technical_report_120515.pdf.

[6] T. Kwartler, Text Mining in Practice with R, John Wiley & Sons Ltd, 2017.

[7] E. Loper and S. Bird, "NLTK: The Natural Language Toolkit," in *ETMTNLP '02 Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, Philadelphia (PA), 2002.

[8] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," *Proceedings of 52Nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations,* vol. 01, no. 01, 2014.

[9] I. G. Richard Eckart de Castilho, "A broad-coverage collection of portable NLP components for building shareable analysis pipelines," in *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, Dublin, Ireland, 2014.

[10] A. K. McCallum, "MALLET: A Machine Learning for Language Toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu.

[11] "NLP4J," [Online]. Available: https://emorynlp.github.io/nlp4j/.

[12] "LingPipe," [Online]. Available: http://alias-i.com/lingpipe-3.9.3/.

[13] "spaCy," [Online]. Available: https://github.com/explosion/spaCy.

[14] M. G. J. N. M. T. O. D. P. L. N. P. M. S. M. Z. L. Gardner, "AllenNLP: A Deep Semantic Natural Language Processing Platform," 2017. [Online]. Available: https://www.semanticscholar.org/paper/A-Deep-Semantic-Natural-Language-

Processing-Gardner-Grus/a5502187140cdd98d76ae711973dbcdaf1fef46d.

[15] "PyTorch," [Online]. Available: https://pytorch.org/. [Accessed 23 08 2019].

[16] "NLP Architect," Intel, [Online]. Available: https://www.intel.ai/introducing-nlp-architect-by-intel-ai-lab.

[17] "flair," [Online]. Available: https://github.com/zalandoresearch/flair.

[18] A. Foundation, "Apache Lucene," [Online]. Available: https://lucene.apache.org/.

XX-NA-XXXXX-EN-N

The European Commission's science and knowledge service

Joint Research Centre

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub

ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

EU Science, Research and Innovation

EU Science Hub

Publications Office of the European Union