2016 IEEE International Conference on Data Science and Advanced Analytics

Senpy: A Pragmatic Linked Sentiment Analysis Framework

J. Fernando Sánchez-Rada, Carlos A. Iglesias, Ignacio Corcuera and Óscar Araque

Intelligent Systems Group
Universidad Politécnica de Madrid
{jfernando,cif}@dit.upm.es, {ignacio.cplatas,oscar.aiborra}@alumnos.upm.es

***Abstract*—Sentiment and emotion analysis technologies have quickly gained momentum in industry and academia. This popularity has spawned a myriad of service and tools. Due to the lack of common interfaces and models, each of these services imposes specific interfaces and representation models. Heterogeneity makes it costly to integrate different services, evaluate them or switch between them. This work aims to remedy heterogeneity by providing an extensible framework and an API aligned with the NLP Interchange Format service specification. It also includes a reference implementation, a first step towards a successful and cost-effective adoption. The specific contributions in this paper are: (i) the Senpy framework; (ii) an architecture for the framework that follows a plug-in approach; (iii) a reference open source implementation of the architecture; (iv) the use and validation of the framework and architecture in a big data sentiment analysis European project. Our aim is to foster the development of a new generation of emotion aware services by isolating the development of new algorithms from the representation of results and the deployment of services.**

Keywords*—*sentiment analysis; emotion analysis; framework; linked data;

I. Introduction

Sentiment analysis is a blooming field of research and application, fueled by the popularity of social media and the need to make sense of collective opinions [1]. A vast number of sentiment analysis tools and services have emerged in recent years. Most of these tools and services use ad-hoc representation and schemas. This heterogeneity not only prevents reusing tools, but it also hinders the establishment of common terminology and models. Initiatives like NLP Interchange Format (NIF) [2] paved the way to standardization by publishing a semantic format and an API for NLP services. Thence, applications like the NIF combinator [3] appeared, demonstrating that a semantic format eases the integration of different services. Other works have applied this notion to multimodal sentiment analysis by extending NIF with existing and new ontologies [4]. The new ontologies for emotion representation enable a better and unambiguous annotation, as well as other interesting applications such as automatic mapping of emotions between different models (e.g. from Plutchik's categories to the Valence-Arousal-Dominance space). However, the concepts behind ontologies and linked data publishing are unfamiliar to both the linguistic community and developers. As a consequence, there are still few solutions in the field that use semantic technologies. This is a known problem that motivated the creation of JSON-LD [5].

The contributions of this paper are: (i) Senpy, a generic framework for NLP services based on the vocabularies NIF, Marl and Onyx; (ii) the architecture of a service in this framework; (iii) the reference implementation of the Senpy architecture, which follows a plug-in architecture and demonstrates the practical feasibility of the framework [6], as well as several plugins for custom algorithms and wrappers of popular services; (iv) the extensive use of the reference implementation in a big data sentiment in the context of a big data sentiment analysis platform and other research projects.

The ultimate goal of this work is to ease the adoption of the proposed linked data model for sentiment and emotion analysis services, so that services from different providers become interoperable. With this aim, the design of the reference implementation has focused on its extensibility and reusability. A modular approach allows organizations to replace individual components with custom ones developed in-house. Furthermore, organizations can benefit from reusing prepackaged modules that provide advanced functionalities, such as algorithms for sentiment and emotion analysis, linked data publication or emotion and sentiment mapping between different providers.

The rest of this paper is structured as follows. Section II introduces the main concepts behind Senpy and its linked data approach. Section III describes the architecture of the Senpy framework. Section IV describes the reference implementation of the framework. Section V illustrates how this architecture and existing tools can be used to develop and use an emotion analysis service; Lastly, Section VI presents our conclusions and future work.

II. Background

A key aspect of Senpy is its linked data approach. Its model is based on the following specifications:

- Marl [7], a vocabulary designed to annotate and describe subjective opinions expressed on the web or in information systems
- Onyx [8], which is built on the same principles as Marl to annotate and describe emotions, and provides interoperability with Emotion Markup Language (EmotionML) [9]
- NIF 2.0 [2], which defines a semantic format and API for improving interoperability among natural language

978-1-5090-5206-6/16 $31.00 © 2016 IEEE
DOI 10.1109/DSAA.2016.79

processing services NIF follows a linked data principled approach so that different tools or services can annotate a text. To this end, texts are converted to RDF literals and an URI is generated so that annotations can be defined for that text in a linked data way. NIF offers different URI Schemes to identify text fragments inside a string, e.g. a scheme based on RFC5147 [10], and a custom scheme based on context. In addition to the format itself, NIF 2.0 defines a REST API for Natural Language Processing (NLP) services with standardized parameters.

The integration of these ontologies has been covered in previous works [4]. For the sake of clarity, Listing 1 provides an example of annotation by a sentiment analysis service. In particular, it consists of the analysis of a microblog post with the text "The example they used was really good, I really enjoyed it" and whose URL is `http://microblog.com/User1/Post1`. The service response shown in Listing 1 indicates that an opinion (`:Opinion1`) has been detected. The properties of the entity are shown as well. Finally, it provides details of the analysis, such as the algorithm used, its confidence, polarity range and provenance (using the PROV-O ontology [11]). Note that a query to the service has the following format: `http://{endpoint}?i={text}&prefix={prefix}`.

```
<http://microblog.com/User1/Post1#char=0,49>
  rdf:type nif:RDF5147String, nif:Context;
  nif:beginIndex "0";
  nif:endIndex "75";
  nif:isString "The example they used in their last paper
      ↪ was very clear, I really liked it";
  marl:hasOpinion :Opinion1.
:Opinion1
  rdf:type marl:Opinion;
  marl:describesObject "paper";
  marl:describesObjectPart "example";
  marl:describesFeature  "clarity";
  marl:polarityValue "0.8";
  marl:hasPolarity: marl:Positive;
  prov:wasGeneratedBy :Analysis1.

:Analysis1
  rdf:type marl:SentimentAnalysis;
  marl:maxPolarityValue "1";
  marl:minPolarityValue "-1";
  marl:algorithm "dictionary-based";
  prov:wasAssociatedWith http://www.gsi.dit.upm.es/.
```

Listing 1: NIF + Marl output of a service call http://senpy.cluster.gsi.dit.upm.es/api?i=The example they used in their last paper was very clear, I really liked it&prefix=http://microblog.com/User1/Post1#

III. Framework

This section describes a framework for natural language processing services, with a special focus on sentiment and emotion analysis.

The main component of a sentiment analysis service is the algorithm itself. However, for the algorithm to work, it needs to get the appropriate parameters from the user, format the results according to the defined API, interact with the user when errors occur or more information is needed, etc. All this boilerplate of sorts, albeit essential for the service, is a burden on service developers. The situation is even worse when dealing with different algorithms at the same time, which usually requires developing and deploying them separately. For this reason, Senpy proposes a modular and dynamic architecture that allows: i) implementing different algorithms in an extensible way, yet offering a common interface, ii) offering common services that facilitate development, so developers can focus on implementing new and better algorithms. Furthermore, it fosters the creation of common tools such as service validators, evaluation suites and testing tools.

The framework covers all the aspects of developing, publishing and using a sentiment analysis service. These aspects are grouped into layers. In addition to giving a clearer view of the components of a service, separating the framework in aspects serves another purpose: it later helps with transferring this modularity to its implementations. Finally, modular implementation fosters the creation of new services and functionalities by reducing the cost of adding new features and algorithms.

As of this writing, we have identified five different layers: the Analysis Layer includes the core NLP process and the libraries to connect it to the rest of the layers; the Semantic Layer deals with conceptual models and their integration; the Syntactic Layer handles issues such as formatting, serialization and input/output validation; the User Interface (UI) is the way in which users interact with services; the Evaluation Layer allows users to benchmark different algorithms; and the Service Administration Layer offers tools and information to control running services. Figure 1 depicts these layers and the main components within each of them. The rest of this section describe each layer in detail.

The Analysis Layer includes those components that are directly involved in generating new annotations for a given input. More specifically, it comprises the implemented analysis algorithms and the libraries used in the implementation that are responsible for integrating one particular algorithm with the rest of the components. For instance, a specific service may include one or more sentiment analysis algorithms to choose from, a Named Entity Recognition algorithm, a gender detection algorithm, etc. Each of these algorithms should be developed independently from the rest, and should contain only the logic that concerns the generation of new annotations. The interface between every algorithm and the rest of the layers is well defined. The set of libraries that implement this interface are the Senpy SDK, which is also part of the framework.

The Semantic Layer provides semantic consistency to the service and adapts the results from the Analysis Layer to every request. To exemplify the role of this layer, let us consider the case of an emotion analysis service. The Analysis Layer of this service would consist of at least one implementation of an emotion analysis algorithm. This algorithm generates annotations using Ekman's six categories. In a traditional service, this would mean that the output of the service could only contain these categories. If an application requires a different representation, such as the VAD (valence, arousal, dominance) dimensional model, the conversion of the results is external to the analysis service. In a Senpy service, the Semantic Layer could include mappings to transparently adapt the annotations to the desired representation. In addition to mappings and conversion, the Semantic Layer could include other steps, such as validation and inference.

The lowest level of abstraction corresponds to the Syntactic Layer. Its role is to validate and adapt input from and output to the user. On the input side, it extracts all the necessary information from every request, and processess it so that it is understood by other layers. If there is an error in the request, such as missing parameters or wrong syntax, this layer communicates it to the user. When the output from other layers is ready to be sent back to the user, the Syntactic Layer formats it using the appropriate is to validate both the input and the output of the service, to process the input so that it can be understood by other layers, and to process the output so that it has the requested format and structure.

The User Interface (UI) Layer handles the interactions between users and the service. The way in which users make requests and receive the results back is different depending on the medium used. For instance, the same underlying analysis could be accessed through a Command Line Interface (CLI) and a web service (Web UI). This difference should be transparent to developers. Hence, the main task of the UI Layer is to gather requests from the user, forward them to the rest of the framework, and then adapt the output to the medium in use. Another element in this layer is the Playground aspect, which will be explained further in Section IV. The main idea behind it is that users want to experiment with new services before integrating them in their workflow or using them programmatically. The Playground is a simple UI that presents users with all available algorithms and options, and guides them through their use.

All previous layers cover functional aspects, i.e. developing a service and allowing users to make requests. The last two layers in this section cover aspects that do not concern users but developers and service administrators.

A key aspect of developing a new analysis algorithm is to evaluate it and compare it to others. The Evaluation Layer contains benchmarking and evaluation tools. Evaluation is facilitated by the fact that the framework imposes a common API. i.e., services of the same type will use the same annotation scheme and will be called in the same way. Using the common API and a set of gold standard corpora, it is possible to evaluate and compare different algorithms. The same concept applies to testing.

Lastly, the Service Administration Layer includes aspects useful to maintain a service and control its lifecycle. Some of its main functions would be: logging, which is used to control execution and find possible errors; resource manager, to control processing, memory and storage consumption; usage statistics, for an overview on how the service is being used; process monitoring, to control what tasks are running and when; and configuration manager, to view and change the parameters used in the service, such as activating or deactivating modules within the service.

IV. Reference Implementation

Providing a reference implementation of the conceptual framework serves three main purposes. Firstly, it allows us to assess the feasibility and completeness of the framework. Secondly, it acts as a showcase of the purpose and the concepts behind the framework. Thirdly, it can be used as a reference or gold standard for future implementations.

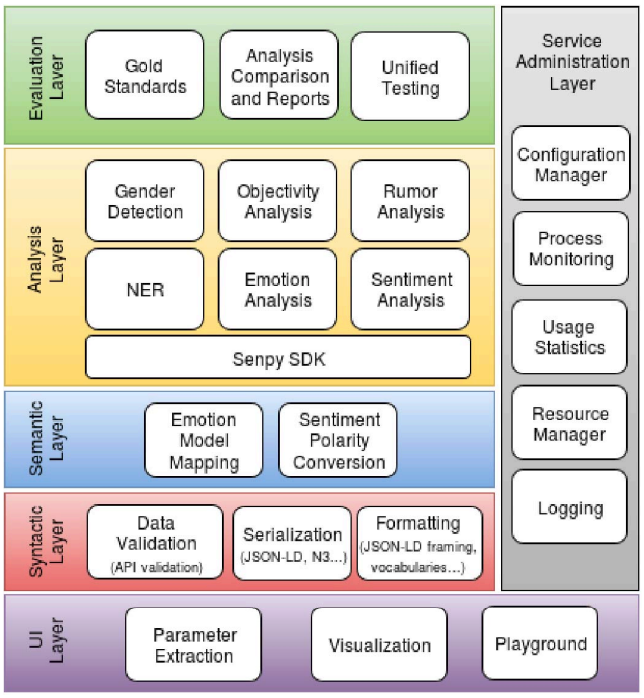

Fig. 1: Senpy framework. Each layer represents a functional block in a service.

The architecture of the reference implementation consists of two main modules: Senpy core, which is the building block of the service, and Senpy plugins, which contain the code for each analysis algorithm. The modularity of the architecture serves the overall goal of Senpy of providing seamless integration of different analysis algorithms while minimizing code duplication and development effort. Several plugins may coexist in the same service, accessing different resources and algorithms while benefiting from the nurturing environment of the common platform. Figure 2 depicts a simplified version of the processes involved in an analysis with Senpy. The following sections describe each component of the architecture in further detail.

The implementation is fully Open Source and published on GitHub[1], and a live demo is publicly available[2].

A. Core

As its name implies, the core of Senpy provides the main functionalities of the platform: an HTTP server/CLI interface, parameter extraction and validation, serialization of results using different formats and an abstraction and publication of results as Linked Data. It manages the lifecycle of plugins as well, orchestrating their execution and all interaction with the user. To better understand the features of the core, let us follow a typical analysis request from a user.

First of all, there are two ways in which a user may want to run their analysis: as a one-off local process or as a service. For

[1] http://www.github.com/gsi-upm/senpy
[2] http://senpy.cluster.gsi.dit.upm.es

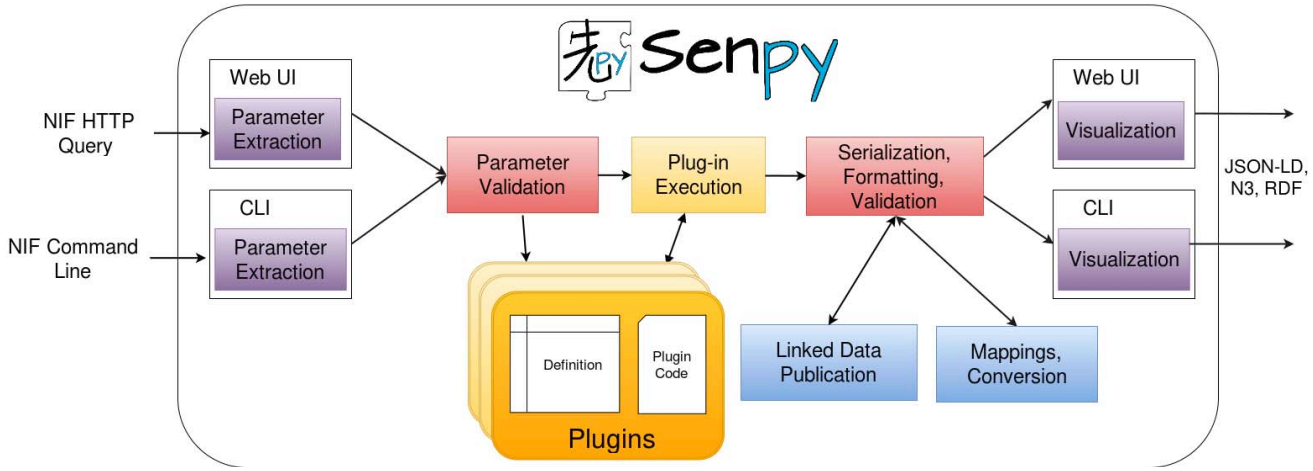

Fig. 2: Modules involved in an analysis with the reference implementation of Senpy

one-off commands, Senpy provides a command line interface (CLI), configurable via arguments. For long running processes or services, Senpy provides an HTTP server. In this case, users send their requests using HTTP queries to the server. Both the CLI and HTTP server use an API aligned with NIF, but using a JSON-LD representation and a JSON-schema by default. This difference makes it friendlier and more appealing to developers, as well as compatible with a wider range of tools. The API defines the parameters that are allowed (Table I), and is complemented by the extra parameters that each plugin declares in its definition (see Listing 2 for an example).

parameter	description
input(i)	serialized data (i.e. the text or other formats, depends on informat)
informat(f)	format in which the input is provided: turtle, text (default) or json-ld
outformat(o)	format in which the output is serialized: turtle (default), text or json-ld
prefix(f)	prefix used to create and parse URIs
emodel(e)	emotion model in which the output is serialized (e.g. WordNet-Affect, PAD, etc.)
minpolarity (min)	minimum polarity value of the sentiment analysis
maxpolarity (max)	maximum polarity value of the sentiment analysis
language (l)	language of the sentiment or emotion analysis
domain (d)	domain of the sentiment or emotion analysis
algorithm (a)	plugin that should be used for this analysis

TABLE I: Parameters of an Emotion or Sentiment analysis service using Senpy

Senpy uses these parameters in every request to extract all parameters from the request, and to warn the user whenever there are missing parameters.

If the basic arguments provided are correct, Senpy uses its selection algorithm to determine the plugin that will receive the request. Typically, users select the plugin manually using the `algorithm` parameter. Senpy will then check if the extra parameters defined in the selected plugin are met as well. If this validation succeeds, the plugin is asked to run an analysis, using the validated parameters.

Senpy leverages different ontologies (e.g. Marl, Onyx) to represent different types of information. For simplicity, the main types of results as well as their required and optional properties have been defined using JSON schema. This means that results are provided in a documented format that can also be validated before passing them to the user. Plugins use these models to return their results.

Once the analysis is done, its results are further modified before they are returned to the user. First of all, values are transformed to fit the parameters specified by the user. For instance, when a plugin uses a sentiment value in the interval (-1, 1), and the user requested a value in the (0,) interval. This phase is very useful when dealing with emotions. Senpy has several mappings from dimensional models to categories, and vice versa. An example of this can be seen in Section V.

Lastly, Senpy generates the final results in the appropriate format, including metadata and proper URI identifiers, so it can be published as Linked Data.

For convenience, Senpy includes a web interface to test all available plugins: the Senpy Playground (Figure 3). The Playground lists all available services, and dynamically adds fields for every parameter they accept, such as language.

B. Plugins

The components in the Analysis Framework from Figure 1 are plugins in the implementation. Hence, each plugin represents a different analysis process. For instance, we may have a plugin for emotion analysis using WordNet-Affect, and a plugin for sentiment analysis using SVM and the Sentiment140 corpus. In future versions of the implementation we plan to extend plugins to also cover components in other layers of the architecture.

A plugin is defined by two elements: a definition file and the plugin code. The definition file can be written in JSON or YAML (a JSON superset), and has the `.senpy` extension. It contains important information about the plugin such as: name, version, location of the plugin code, parameters needed and attributes of the plugin. Listing 2 shows the description file of an example plugin, which we will use in Section V. In this description we see that the plugin accepts an extra parameter

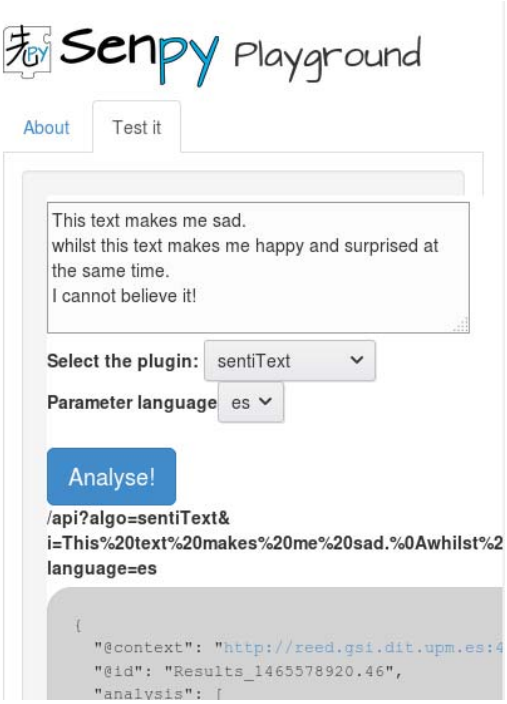

Fig. 3: Senpy Playground web interface.

in requests, `language`. When not provided, this parameter defaults to `en`. It also contains an attribute specific to this plugin, `default_value`, which determines the default value for words that are not found in ANEW.

Listing 2: Plugin definition using YAML

```
---
name: EmoTextANEW
module: emotextANEW
version: "0.1"
description: "Emotion classifier using rule-based
    ↪ classification."
extra_params:
language:
aliases:
- language
- l
default: en
options:
- es
- en
required: true
default_value: [0,0,0]
```

The `module` attribute indicates the module that will be loaded. In this case, that module corresponds to the code in Listing 3. A Senpy (Senpy) plugin has to implement three methods: `activate`, for allocation of resources; `deactivate` for their release; and `analyse`, which takes the user-supplied parameters and performs the analysis. Resource allocation may seem needlessly complicated, but it is an important process when dealing with models that take gigabytes of memory. Section V covers the creation of a this specific plugin in more detail.

There are three main states in the lifecycle of a plugin: unloaded, inactive and active. Only active plugins can be used in requests. For a plugin to be active, two things have to happen. First, the core has to load it. Once a plugin has been loaded, it gets in the inactive state. In this state, a plugin is listed by the core, but the variables necessary for analysis may not have been initialized. When a plugin is loaded, a special method in the plugin is called that initializes these variables. If the activation process is successful, the plugin enters the active state and can be used by users. If there are errors during the activation, the plugin remains inactive and all errors are logged. Active plugins can also be deactivated, which puts them in the inactive state again and should free up any variables that were initialized during activation.

To exemplify this process, let us consider the case of a sentiment analysis that uses a naive bayes classifier. This plugin requires a trained classifier to analyze text. However, when the plugin is loaded the classifier is not ready yet. The classifier is trained upon activation. Training may take a long time, depending on the size of our corpus and the features used. For this reason, changes of state are asynchronous operations for the core. When the activation finishes, our plugin will be automatically marked as active. Meanwhile, the core may handle other requests. Once our plugin is active, we can use it to analyze text. When our plugin is no longer needed, we may deactivate it. Deactivation will free up the memory used by the trained model.

Releasing resources when a plugin is not needed means that many resource hungry plugins can be loaded at the same time, and only activate them when they are needed. Resource initialization during activation also means that plugin variables will be consistent after it is activated. On the other hand, it also means that costly operations, such as training a model, have to be repeated several times. To avoid repetition and speed up start-up time considerably, Senpy ships with a special type of plugins that provide persistence. These persistent plugins have access to special variables that can be used to store the results of costly operations. When these variables are used, the plugin automatically checks the filesystem for a saved version of the variable. If if does, it loads the variable. If not, the plugin runs the appropriate operation and stores the value of the result in the filesystem.

The reference implementation Senpy has been validated by implementing wrappers to several available sentiment and emotion services, such as Sentiment 140 [12], MeaningCloud [13], Cogito [14], Vader [15] and Paradigma [16].

V. Use Case

In this section we briefly cover the process of using Senpy in a real scenario. Our use case is a Big Data platform that uses a series of NLP services on social media. In fact, the scenario is a simplification of one of the pilots in the MixedEmotions project. This platform is made up of several modules from different parties. Some of them are existing NLP and sentiment analysis services. The rest of the modules depend on one or more of these analysis services. Integrating the different modules and their interfaces would require a big effort from every parties involved. Senpy reduces the cost of integration with its common interface and tools.

Figure 4 depicts the main elements. There are two parts in the platform of this use case. Firstly, there is a live brand monitoring dashboard. The dashboard shows the opinions of

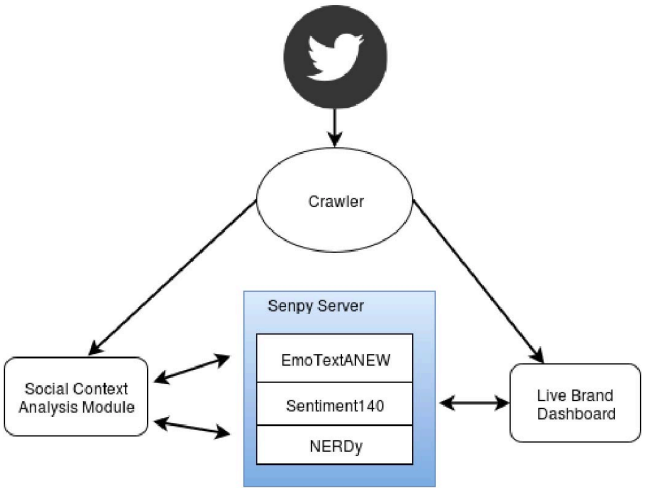

Fig. 4: Using Senpy in a Big Data Sentiment Analysis Platform

social media users about a brand. For this, it uses an external sentiment analysis service (sentiment140 [3]), to annotate social media content with opinions. Secondly, there is a social context analysis module that finds the most influential users and content, as well as the evoluation of emotion of all relevant users. The social context analysis module uses a NER (named entity recognition) module to gather only relevant content, and an emotion analysis module to annotate the emotions in the content. Social media content is provided by a separated module, labeled Crawler in Figure 4. A server running senpy will provide the NER, Sentiment and Emotion Analysis Analysis services.

Instead of accessing the external sentiment analysis service directly, we choose to use a custom sentiment analysis service in senpy that acts as a wrapper/proxy to the actual service. The main advantage of this approach is that it avoids having to deal with more than one API and schema for NLP service. Additionally, we gain access to all the extra capabilities of Senpy, such as the polarity conversion, benchmarking and service evaluation. Developing the wrapper is very straightforward and requires very little code [4].

The emotion analysis analysis relies on the ANEW [17] lexicon to analyze the emotion in text, using a simple bag-of-words approach. Turning this code into a Senpy plugin is trivial, and merely a matter of implementing the `analyse` method. We have already covered the description file in Listing 2. The accompanying code is shown in Listing 3.

Listing 3: Code for the EmoTextANEW plugin

```
from os.path import dirname, abspath
from senpy.plugins import SenpyPlugin, tokenize
from senpy.models import Results, EmotionSet,
                         Emotion, VAD
from emotextanew import Analyser

class EmoTextANEW(SenpyPlugin):

    def activate(self, *args, **kwargs):
        self._local_path = dirname(abspath(__file__))
        self._analyser = Analyser(self._local_path)

    def deactivate(self, *args, **kwargs):
        del self._analyser

    def analyse(self, params):
        r = Results.from_params(params)
        for i in r.entries:
            es = EmotionSet()
            e = Emotion()
            valence = 0
            arousal = 0
            dominance = 0
            for j in i.nif__isString:
                # Find V,A,D for each word
                v, a, d = self._analyser.get_vad(j)
                valence += v
                arousal += a
                dominance += d

            e[VAD.arousal] = a
            e[VAD.valence] = v
            e[VAD.dominance] = d
            es.onyx__hasEmotion.append(e)
            i.emotions = es
        return results
```

Since ANEW uses the VAD emotion model, that is what our plugin will use as well. Normally, this would mean users would need to use the VAD model themselves. However, since we are using Senpy to publish our service, we can make use of its additional features, such as mapping of emotion models. Emotion mapping can be used by setting the `emodel` parameter in the request. Listing 4 shows the response to a request using the WordNet-Affect model. Notice the addition of the emotion category (joy) based on the VAD dimensions. In particular, the conversion from VAD to WordNet-Affect categories is based on centroids [18]. The information about the centroids is displayed in the results, which together with the use of the provenance ontology makes the process transparent and repeatable.

Listing 4: Requesting an emotion analysis with a different emotion than the one provided in the plugin.

```
{
  "@context": "http://senpy.cluster.gsi.dit.upm.es/api/
      ↪ contexts/context.jsonld",
  ]
  "analysis": [
  {
 "analysis": [
   {
     "@id": "EmoTextANEW_0.1",
     "@type": "onyx:EmotionAnalysis",
     "onyx:usesEmotionModel": "onyx-anew:ANEWModel"
   },
   {
     "@id": "ANEW_Mappings_0.1",
     "@type": "onyx:EmotionAnalysis",
     "onyx:usesEmotionModel": "wnaffect:WNAModel"
     "centroids": {
       "wnaffect:anger": {
         "A": 6.95,
         "D": 5.1,
         "V": 2.7
       },
       "wnaffect:disgust": {
         "A": 5.3,
         "D": 8.05,
         "V": 2.7
       },
       "wnaffect:fear": {
         "A": 6.5,
         "D": 3.6,
         "V": 3.2
       },
       "wnaffect:joy": {
         "A": 7.22,
```

[3] http://www.sentiment140.com/

[4] https://github.com/gsi-upm/senpy/tree/master/senpy/plugins/sentiment140

```
        "D": 6.28,
        "V": 8.6
      },
      "wnaffect:sadness": {
        "A": 5.21,
        "D": 2.82,
        "V": 2.21
      }
    }
  }
  ],
  "entries": [
  {
    "emotions": [
    {
      "onyx:hasEmotion": [
      {
        "prov:wasGeneratedBy": "EmoTextANEW_0.1",
        "onyx-anew:arousal": 5.0,
        "onyx-anew:dominance": 5.62,
        "onyx-anew:valence": 6.23,
      },{
        "prov:wasGeneratedBy": "ANEW_Mappings_0.1",
        "onyx:hasEmotionCategory": "wn-affect:joy"
      }]
    }
    ],
    "language": "en",
    "nif:isString": "I am feeling excited"
  }
 ]
}
```

This section illustrates how easy it is to develop a new service from scratch and to integrate it in a bigger scenario. As shown in Listing 3, a plugin code is made up entirely of the `analyse` function, which almost perfectly matches the pseudocode of the algorithm. This succinct code provides a web service and a CLI tool that does parameter extraction and validation automatically. Furthermore, the platform provides additional features such as automatic linked data conversion and publication or mapping of emotions. Finally, the common interfaces and schemas provide loose coupling to the platform. This means that once a module is adapted for senpy to make use of a service, it can be made to use any equivalent service just by pointing it to a different endpoint.

VI. Conclusions and future work

The sentiment and emotion analysis community would highly benefit from a common framework for service and language resources. Such a framework would ease adoption, development, integration and evaluation of services. A linked data approach further adds to these benefits, but its use needs to be transparent to users and developers.

This paper proposes a generic framework that combines both worlds and a reference implementation of that framework that is currently being used in MixedEmotions[5], a European R&D project. The linked data model for sentiment and emotion services is based on the combination of NIF, Marl and Onyx vocabularies. Moreover, a number of parameters (e.g. min, max and e) have been defined following NIF Service specification so that sentiment and emotion service calls are interoperable.

We want to increase the adoption of the framework and to foster a community approach, where most plugins and features are provided by third parties. For this reason, we are currently working on easing the development of new plugins, and on making it possible to create plugins for any part of the framework. Other lines of research would be the connection between different plugins (e.g. pipelining), the integration of the framework with other distributed and big data systems, and the addition of authentication and rate limiting.

In conclusion, the framework proposed in this paper has already proven useful in a multilingual sentiment analysis scenario. It has enabled the integration of multiple services from different parties and eased the creation of novel algorithms. This new approach paves the way for new testing and validation tools, as well as advanced capabilities such as deployment in high availability and cluster environments.

[5]http://mixedemotions-project.eu/

VII. Acknowledgements

This work has been partially funded by the European Union - Horizon 2020 (Industrial Leadership) through the MixedEmotions project (number H2020 644632). Part of this work has been carried out in the context of the MOSI-AGIL-CM research programme (grant S2013/ICE-3019, supported by the Autonomous Region of Madrid and co-funded by EU Structural Funds FSE and FEDER, and the SEMOLA project, funded by the Ministry of Economy and Competitiveness of Spain (TEC2015-68284-R).

References

[1] B. Liu, "Sentiment analysis and opinion mining," *Synthesis Lectures on Human Language Technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[2] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer, "Integrating nlp using linked data," in *The Semantic Web–ISWC 2013*. Springer, 2013, pp. 98–113.

[3] S. Hellmann, J. Lehmann, S. Auer, and M. Nitzschke, "Nif combinator: Combining nlp tool output," in *Knowledge Engineering and Knowledge Management*. Springer, 2012, pp. 446–449.

[4] J. F. Sánchez-Rada, C. A. Iglesias, and R. Gil, "A linked data model for multimodal sentiment and emotion analysis," *ACLIJCNLP 2015*, p. 11, 2015.

[5] M. Lanthaler and C. Gütl, "On using json-ld to create evolvable restful services," in *Proceedings of the Third International Workshop on RESTful Design*. ACM, 2012, pp. 25–32.

[6] E. Dalci, E. Fong, and A. Goldfine, "Requirements for gsc-is reference implementations. national institute of standards and technology," *Information Technology Laboratory*, 2003.

[7] A. Westerski, C. A. Iglesias, and F. Tapia, "Linked opinions: Describing sentiments on the structured web of data," in *Proceedings of the Fourth International Workshop on Social Data on the Web (SDoW2011)*. CEUR, Oct. 2011, pp. 21–32.

[8] J. F. Sánchez-Rada and C. A. Iglesias, "Onyx: A linked data approach to emotion representation," *Information Processing & Management*, vol. 52, no. 1, pp. 99–114, 2016.

[9] M. Schröder, P. Baggia, F. Burkhardt, C. Pelachaud, C. Peter, and E. Zovato, "Emotionml – an upcoming standard for representing emotions and related states," in *Affective Computing and Intelligent Interaction*, ser. Lecture Notes in Computer Science, S. D'Mello, A. Graesser, B. Schuller, and J.-C. Martin, Eds. Springer Berlin Heidelberg, 2011, vol. 6974, pp. 316–325.

[10] E. Wilde and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type," Internet Engineering Task Force, Apr. 2008.

[11] P. Missier, K. Belhajjame, and J. Cheney, "The w3c prov family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 2013, pp. 773–776.

[12] P. Chikersal, S. Poria, and E. Cambria, "Sentu: sentiment analysis of tweets by combining a rule-based classifier with supervised learning," in *Proceedings of the International Workshop on Semantic Evaluation, SemEval*, 2015, pp. 647–651.

[13] I. Segura-Bedmar, P. Martínez, R. Revert, and J. Moreno-Schneider, "Exploring spanish health social media for detecting drug effects," *BMC medical informatics and decision making*, vol. 15, no. Suppl 2, p. S6, 2015.

[14] Expert System, *COGITO Intelligence Platform*, 2015.

[15] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[16] J. F. Sánchez-Rada, M. Torres, C. A. Iglesias, R. Maestre, and R. Peinado, "A Linked Data Approach to Sentiment and Emotion Analysis of Twitter in the Financial Domain," in *Second International Workshop on Finance and Economics on the Semantic Web (FEOSW 2014)*, vol. 1240, May 2014, pp. 51–62.

[17] M. M. Bradley and P. J. Lang, "Affective norms for english words (ANEW): Instruction manual and affective ratings," Technical Report C-1, The Center for Research in Psychophysiology, University of Florida, Tech. Rep., 1999.

[18] S. M. Kim, A. Valitutti, and R. A. Calvo, "Evaluation of unsupervised emotion models to textual affect recognition," in *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. Association for Computational Linguistics, 2010, pp. 62–70.