Heracles: a Framework for Developing and Evaluating Text Mining Algorithms

Kim Schouten, Flavius Frasincar*, Rommert Dekker, Mark Riezebos

Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, the Netherlands

Abstract

Many of today's businesses are driven by data, and while traditionally only quantitative data is considered, the role of textual data in our digital world is rapidly increasing. Text mining allows to extract and aggregate numerical data from textual documents, which in turn can be used to improve key decision processes. In this paper, we propose Heracles, a framework for developing and evaluating text mining algorithms, with a broad range of applications in industry. In contrast to other frameworks, Heracles supports both the development and evaluation stages of text mining algorithms. A practical use case shows the versatility and ease-of-use of the proposed framework in the domain of aspect-based sentiment analysis.

Keywords: text mining, algorithm evaluation, research and development, developers framework

1. Introduction

With text mining becoming ever more popular, numerous companies, labs, and research groups are committed to developing the next generation of text mining algorithms (Aggarwal and Zhai, 2012). As textual data abound on the Web, there are many applications in which text mining plays a key role. A prolific example of text mining is sentiment analysis on financial news, a topic that is popular in both academia (Schumaker et al., 2012; Chan and Chong, 2017), and business (Bloomberg, 2018), due to its challenging nature and potential profitability. Another use of text mining can be found in modeling and managing customer satisfaction (Takeuchi et al.,

*Corresponding author; tel: +31 (0)10 408 1340

Email addresses: `schouten@ese.eur.nl` (Kim Schouten), `frasincar@ese.eur.nl` (Flavius Frasincar), `rdekker@ese.eur.nl` (Rommert Dekker), `riezebos@ese.eur.nl` (Mark Riezebos)

2009; Farhadloo et al., 2016), a topic that is all the more relevant now that many major companies have social media divisions that monitor and engage with customers on various platforms. Some social media, such as Twitter, offer a unique possibility to harness the power of millions of personal expressions for problems ranging from predicting soccer matches (Schumaker et al., 2016) to natural disaster management (Spence et al., 2015). Social emotion mining (Rao et al., 2014), or gauging the emotional response of a target group of online users, is also a particularly compelling use case.

While in academia, there are mature evaluation practices that ensure scientifically valid performance reports, these practices are not as ubiquitous in business, where focusing on rapid prototypes, fast deployment, and customer satisfaction is often more important than measuring the exact performance of the developed product. Even so, we argue that for academics and business researchers alike, using a solid evaluation methodology is of great benefit. This is the main reason we developed a framework that supports both the development and the evaluation of text mining algorithms. Since text mining is at the intersection of machine learning and natural language processing, support is needed in the form of machine learning algorithms that can be used to solve text mining problems, as well as natural language components to draw upon when processing the raw, unstructured text. Hence, the advantage of Heracles is that it supports all three parts of the process: natural language processing to deal with the raw textual data, machine learning algorithms to perform the required text mining tasks on the processed data, and evaluation procedures to assess the performance of a developed algorithm. While many frameworks exist that address one or two of the previously mentioned parts, we argue that all three are needed for a system to be truly beneficial to text mining developers.

The framework proposed in this paper is born out of the need to develop and test algorithms for sentiment analysis, one of the more popular branches of text mining (Feldman, 2013). For our participation in the sentiment analysis task at the well-known benchmarking workshop SemEval, we incorporated the used evaluation methodology, which is the de facto standard for evaluating text mining algorithms, in our software. SemEval (Kilgariff and Palmer, 1998; Bethard et al., 2016) is a long running workshop where current problems in text mining are targeted by releasing an annotated data set and issuing a challenge for researchers to compete and build the best algorithm for the given task. Tasks range from fine-grained sentiment analysis to semantic taxonomy enrichment. Going through a series of generalization steps to accommodate for different text mining tasks on data sets other then just the SemEval data, we arrived at a unified, powerful suite of tools that will be of use to any developer wishing to create the next successful application in the text mining industry.

This paper aims to describe the design principles of our proposed framework, as well as a number of practical use cases to illustrate its benefits. While we implemented the framework in the Java programming language, including a specific set of built-in tools tailored to our specific needs, the design patterns and architecture presented in this paper are general enough to be implemented in any object-oriented programming language. With this in mind, we included technical details in the system description to aid in this endeavor. A Java implementation of the framework is available at `https://github.com/KSchouten/Heracles`.

The paper is structured as follows. In Section 2 we present the related work by giving an overview of the existing frameworks for developing and evaluating text mining algorithms. Section 3 describes the design principles and architecture of the proposed Heracles framework. A complete use case is given in Section 4, showing the versatility and ease-of-use of the proposed framework. This section also includes an overview of the strengths and weaknesses of the framework which are derived from the conducted user study. Section 5 is the last section of this paper and contains the conclusions as well as suggestions for future work.

2. Related Work

There are many software frameworks (i.e., workbenches, toolkits, libraries, packages, etc.) for text mining, both open and proprietary, and we can name only a few in this section. Because of the closed nature of proprietary systems we are limited to discussing only open and freely available frameworks. All the investigated frameworks are listed in Table 1, along with some characteristics for each entry. The three main dimensions we use to evaluate existing frameworks are (1) their ability to support the natural language processing (NLP) part of developing a text mining algorithm, (2) their ability to support the machine learning (ML) part of developing a text mining algorithm, and (3) their ability to promote and enable rigorous evaluation of the developed algorithms. Furthermore, we look at whether a graphical user interface (GUI) or application programming interface (API) is available, whether parallel processing is supported, whether multiple input formats are possible, and, for NLP packages only, whether support for multiple languages is implemented. Last, we check if there are certain requirements for using the software, such as having the Java Virtual Machine running.

Many software packages integrate or provide wrappers for other pieces of software. This can confuse the license under which the software is made available. Hence, the license column in Table 1 refers to the license under which the core software is made available. Nevertheless, using third-party libraries will usually invoke the license of that component. For example, NLTK is licensed under the Apache license, which gives users a lot of freedom to use and modify the software, including commercial use. However, linked components might have their own license, so using, e.g., CoreNLP from within NLTK will invoke the stricter GPL license that CoreNLP uses.

The discussed software packages are divided into four groups. The first group consists of software that mainly provides NLP functionality. The second group is formed by software packages that provide support for machine learning algorithms. The third category is a singleton group, containing Gerbil, an independent evaluation-only Web framework. The last group is a set of frameworks that combine ML and NLP to support the development of new algorithms, in a fashion similar to our proposed Heracles framework. Last, we will present a brief feature overview of Heracles, to put it in perspective with the other discussed software packages.

2.1. Natural Language Processing Software

Currently, there are various software packages available that provide extensive NLP functionality. One well-known and widely used library is the Stanford CoreNLP (Manning et al., 2014). It can perform various NLP tasks like tokenization, part-of-speech tagging, lemmatization, but also named entity recognition, sentiment analysis, and grammatical parsing. It is written in Java, but it also has a server setup, where one can use a Web API to access its features from any language. CoreNLP is also integrated in several other software packages and frameworks, illustrating its positive qualities. CoreNLP can easily process sentences in parallel by setting the appropriate option

	GUI	API	NLP Development	Multiple Languages	ML Development	Evaluation	Parallel Computing	Multiple Input Formats	License	System Requirements
Stanford CoreNLP	*	V	V	V	-		V		GPL v3	Java
NLTK	-	V	V	V	V	*	*		Apache v2	Python
OpenNLP	-	V	V	V	*	V	*		Apache v2	Java
xTAS	-	V	V	V	-		V	-	Apache v2	Linux, Python, and Java
Weka	V	V	-		V	V	V	V	GPL v3	Java
Scikit-learn	-	V	-		V	V	V	V	BSD	Python
TensorFlow	-	V	-		V	*	V	V	Apache v2	Python
Apache Spark	-	V	-		V	V	V	V	Apache v2	Java
Apache Mahout	-	V	-		V	*	V		Apache v2	Java
Mallet	-	V	-		V	V	*	V	CPL v1	Java
Watchmaker	*	V	-		V	-	V		Apache v2	Java
Java-ML	-	V	-		V	V	-	V	GPL v2	Java
GERBIL	V	V	-		-	V		-	LGPL v3	public REST interface
GATE	V	V	V	V	*	*	-	V	LGPL v3	Java
LingPipe	-	V	V	V	V	V	*	-	custom	Java
@Note	V	V	V	*	V	V	-	V	GPL v3	Java
CLULAB	-	V	V	*	V	V	*		Apache v2, GPL v3	Scala
Heracles	**-**	**V**	**V**	**V**	**V**	**V**	**V**	**V**	**GPL v3**	**Java**

Table 1: An overview of existing software packages and some of their characteristics.

to the number of threads that can be used. While licensed under the strictly open source GPL license, commercial licenses are available upon request.

The Apache OpenNLP (Apache OpenNLP, 2018) project has a different take on NLP by providing the developer with both pre-trained models as well as an easy way to train and evaluate the models on custom data. All algorithms are internal to the library instead of linking to other software packages. On the other hand, the NLTK (Bird et al., 2009) package aims to provide a full range of tools for NLP, and includes links to other packages to extend its functionality, such as Weka and CoreNLP. It also comes with a number of basic machine learning algorithms a developer can use to build a custom algorithm. However, advanced evaluation functionality like the cross-validation are absent. Both OpenNLP and NLTK do not explicitly provide parallel processing capabilities, although key components are thread-safe, enabling the developer to use them in a custom parallel computing setup. NLTK can be parallelized with third-party libraries, such as Execnet (Krekel, 2018), which uses the Message Passing Interface to safely communicate between processes that otherwise do not share any resources. Both OpenNLP and NLTK are licensed under the Apache license, giving the developer full freedom in how to use these packages.

A general text mining framework called xTas is proposed in (de Rooij et al., 2012), that includes both wrappers for popular packages as well as custom-built modules coming out of research. It uses Elasticsearch for distributed document storage and retrieval. While it is possible to run xTas locally as a Python module, it can also be run as a service, distributing tasks over multiple worker processes as necessary. Developers are encouraged to extend xTas by defining custom tasks, which will automatically run in a distributed fashion within the xTas framework. The xTas core software is licensed under the liberal Apache license, but several wrappers call upon software that is under stricter licenses. The system is developed for python, but currently runs only on linux operating systems. Java is required to use some of the provided wrappers, such as the one for CoreNLP.

2.2. Machine Learning Software

Next to the previously discussed NLP solutions, there are multiple frameworks for machine learning (ML), each with its own characteristics. A well-known framework in the scientific community is Weka (Witten and Frank, 2005), a general purpose, Java-based, machine learning framework that comes packaged with many algorithms. It provides proper evaluation support and includes also methods for feature selection and meta-optimization. Weka has specialized classes that can be used to utilize parallel computing, meaning that by default it will be single threaded. Its popularity is illustrated by the fact that various frameworks include wrappers for Weka (e.g., (Abeel et al., 2009; Cunningham et al., 2011)). An extra benefit is that Weka has a full-fledged graphical user interface where all implemented algorithms can be run over a dataset, which can be in any of the supported file formats. While no new algorithms can be developed in the graphical interface, it makes the existing ones accessible to a more general public. Weka itself is licensed under the GPL license, with various algorithms under different licenses, but commercial licenses may be available upon request.

Scikit-Learn (Pedregosa et al., 2011) could be considered the Python counterpart to Weka. It has multiple algorithms for the various machine learning tasks, such as classification, regression, and clustering. It is built on the popular NumPy, SciPy, and matplotlib libraries and is licensed under the liberal BSD license, which also allows commercial use. It can read files in the libsvm (Chang and Lin, 2011) format, as well as anything that is compatible with NumPy arrays or SciPy sparse matrices. Scikit-Learn has the permissive BSD license.

Also for Python is the Google TensorFlow (Abadi et al., 2015) project, a framework for deeply learned neural networks. It has APIs for various languages besides Python although not every API is as stable and as complete. TensorFlow supports parallel processing, both on muliple CPUs and multiple GPUs with CUDA (Nickolls et al., 2008) support. While TensorFlow can evaluate the performance of a trained model, more advanced methods like support for cross-validation is not available. TensorFlow is licensed with the Apache license, allowing a full range of application.

Other frameworks that explicitly support parallel processing are Apache Spark (Zaharia et al., 2016) and the older Apache Mahout (Lyubimov and Palumbo, 2016). As Apache projects, both software packages have the Apache license. While Spark is a general purpose cluster computing framework, its built-in MLib (Meng et al., 2016) library provides a good number of machine learning algorithms that can be used in custom applications. In the past, Mahout used to run on Hadoop MapReduce (White, 2009), but nowadays it supports different engines, including the much faster

Apache Spark. One of its main attractive features is its ability to perform linear algebra on big data sets.

For Java, another mention is Mallet (McCallum, 2002), which provides machine learning algorithms that are widely used in language processing. For example, it provides an implementation of Latent Direchlet Allocation (Blei et al., 2003) and graphical models in general, which is absent from, e.g., Weka. Sequence tagging is another area where Mallet complements Weka, by providing an implementation of Conditional Random Fields (Lafferty et al., 2001). Mallet is licensed under the CPL, a precursor to the Apache license that is similar in nature to the latter.

The Watchmaker (Watchmaker, 2018) framework provides evolutionary computing, like genetic algorithms, that can be used to optimize an arbitrary algorithm. The developer will need to translate the parameters to be optimized into a numeric parameter array and a specific function to compute the performance, given some input, needs to be implemented as well. Watchmaker will then be able to change, or evolve, the parameter array and get the performance for each. Keeping the good ones around and combining them into a new generation of possible solutions, etc., will result in a good set of parameters. Since this is a heuristic method, an optimal result cannot be guaranteed. While Watchmaker does not have a full graphical user interface, it does provide UI elements that can be integrated in a custom application that uses Watchmaker. For instance, a graphical tracker of the population, generations, and performance can be used to show progress of the optimizer. Watchmaker can be incorporated in all kinds of software systems, as it is distributed under the Apache license.

As an honorable mention, we would like to list Java-ML (Abeel et al., 2009) here too, as it is one of the earlier Java frameworks for machine learning. It includes wrappers for Weka, making it easy to use any of its machine learning implementations. Other strong points include unified interfaces for each type of algorithm, and many examples and reference implementations.

2.3. Independent Evaluation Framework

A system that has scientific testing at its core is GERBIL (Usbeck et al., 2015). It successfully abstracts away from specific algorithms by specifying a common REST interface any algorithm should adhere to for using this framework. All algorithms implementing this interface can be easily linked up with the core benchmark module. Data sets are handled in a similar way, being stored in a different location (i.e., DataHub in this case). While GERBIL is primarily designed for the entity annotation task, the concept should apply to any task that produces annotations in text, which is the case for practically all text mining problems. A strong point of the framework is that experiments done using the REST API are permanently stored using a persistent URL, so previously performed experiments are logged and can be accessed and redone at the click of a button. In our current day and age where reproducibility is much talked about but often still a near impossible task, this is a great feature to have.

2.4. Text Mining Frameworks

In the text mining field, GATE (Cunningham et al., 2011) (General Architecture for Text Engineering) is a well established software package. With its GUI, it provides an easy way of building a custom NLP pipeline, stringing various text processing components together as needed. By means of various plugins it is possible to perform NLP tasks on text in languages other than English. It is also possible to create custom components for such a pipeline using the API, thereby extending the framework. When gold data is given (Biemann and Mehler, 2014), GATE can evaluate the created annotations, providing a graphical overview of which annotations were incorrect next to the standard performance metrics. By means of the Learning Framework Plugin, it also provides support for machine learning components. In particular, the plugin provides wrappers for machine learning software, such as Mallet (McCallum, 2002), libSVM (Chang and Lin, 2011), parts of Weka (Witten and Frank, 2005), and Scikit-Learn (Pedregosa et al., 2011). Using more elaborate evaluation schemes such as k-fold cross-validation is not included in GATE. Its LGPL license allows developers to link their software to the GATE libraries without having to publish their own source code.

Another, well-known, framework in the field of text mining and NLP is LingPipe (Alias-i, 2018), which supports a variety of NLP and ML tasks. Examples include named entity recognition, topic modeling, classification, basic sentiment analysis, and spelling correction. It also supports the evaluation of algorithms, providing performance measures as well as options such as cross-validation. LingPipe does not provide built-in support for parallel computing, but it has thread-safe models and decoders for concurrent-read exclusive-write synchronization. While not licensed under a general license, it provides a royalty free license, not unlike the GPL, as well as various commercial licenses.

A framework called @Note, targeting the biomedical domain, is presented in (Lourenço et al., 2009). It encapsulates external libraries, such as GATE and Weka, and provides easy access to biomedical data by crawling the PubMed bibliographic catalogue (PubMed, 2018). It includes some functionality for model validation. In terms of architecture, @Note is built on top of a Java application development framework called AIBench (Fdez-Riverola et al., 2012), and hence inherits its patterns, such as the Model-View-Controller pattern. Furthermore, it is a modular framework allowing developers to build plugins for the system. @Note does not seem to support multiple languages by default, but given its link to GATE, which is multilingual, it might be possible to perform text mining tasks in other languages by building a plugin. Unfortunately, @Note does not support parallel processing, but it is completely open source.

The Computational Language Understanding Lab (CLULAB) at the University of Arizona has written several pieces of software around natural language processing and text mining. Their `processors` project contains components for event extraction and Rhetorical Structure Theory discourse parsing (Surdeanu et al., 2015), but also wrappers for Stanford's CoreNLP and Malt-Parser (Nivre et al., 2007). Furthermore, it contains implementations for many common machine learning algorithms, supporting both classification and ranking tasks. The software is written in Scala under the Apache license, with the exception of the CoreNLP wrapper, which is under the GPL due to CoreNLP, which is integrated in that component, being under GPL. While CLULAB does not provide parallel processing support, the creators have made an effort to make the software thread-safe. Some of the provided components support multiple languages, but others, for instance the discourse parsers, do not.

The proposed Heracles framework, has been developed over the past four years to support both educational and research efforts at the Erasmus University Rotterdam. It does not have a graphical user interface, but focuses solely on providing a development environment to create new text mining algorithms. To that end, it supports various natural language processing components, mostly by wrapping the Stanford CoreNLP (Manning et al., 2014) package. Hence, it has the same support for multiple languages as CoreNLP does. Machine learning is supported by incorporating the Weka (Witten and Frank, 2005) toolkit. In that capacity, it is possible for developers to draw upon the parallel computing options that Weka provides. Differently than Weka, Heracles targets a specific class of machine learning algorithms, i.e., the ones related to text mining. Heracles supports the easy building and evaluation of text mining algorithms, as well as importing and saving results in commonly used data formats as JSON and XML. Heracles makes use of some of the Weka packages but provides an easy-to-use layer on top of these for performing text mining. The Heracles framework itself does not provide specific features for parallel computing, but its components are thread-safe. Heracles can also be linked to the Watchmaker (Watchmaker, 2018) framework to provide evolutionary computing capabilities. Evaluation of algorithms is supported in various forms, including the basic random split into training and test set, but also cross-validation is available. Multiple algorithms can be run in sequence and tested against each other for statistically different results using a 2-sided paired t-test. Heracles supports a number of different input formats, including raw text, JSON, and XML formats. For specific data set formats, this can be tweaked so existing annotations are correctly loaded. Heracles is written in Java and due to wrapping CoreNLP, is licensed under GPL.

3. Design and Architecture of Heracles

In this section, we describe Heracles, our proposed framework and discuss its design principles and overall architecture. With the overview of related work in mind, the goal is to present a framework that supports the development process of new text mining algorithms, including evaluating their performance.

When designing any framework, the target audience is one of the key concepts that determines the design choices. For the framework we developed, the targeted group of users is developers. For developers, the big advantage is that existing NLP and ML methods are easily available when developing a custom text mining algorithm, with the whole process of development and testing being performed within a streamlined evaluative workflow. This is achieved by providing boilerplate code and templates that ensure the developer can utilize the workflow, while still providing flexibility to work on different types of text mining algorithms.

There are three major classes of algorithms that Heracles is designed to support. The first is *classification*, where a given word or sequence of words has to be labeled with a value from a preset list of labels. Examples of this class of algorithms are word sense disambiguation, and positive vs. negative sentiment analysis. The second class of supported algorithms is *regression*, where a given word or sequence of words has to be labeled with a numerical, real value. Examples of regression in text mining are predicting sentiment as a value between -1 and 1, or predicting stock prices. The third class is *sequence labeling*, which is the problem of finding and labeling sequences of words within a text. Examples of this are named entity recognition, and explicit aspect detection for sentiment analysis, but also the basic NLP task of splitting a text into sentences can be considered sequence labeling.

The workflow, as supported by the framework is illustrated in Figure 1. It starts by reading a certain raw dataset and performing various NLP tasks on that dataset. The processed dataset is then optionally saved in a JSON format that preserves all the information as computed by the NLP components, so that the often slow NLP procedure does not have to be repeated with every run. Then the developer sets up an experiment, assigns it the processed data, adds the empty algorithm that will be developed, possibly also adds existing algorithms to compare against, and specifies the evaluation conditions (e.g., cross-validation or not). Now the developer is ready to start working on the algorithm, and, while in the development process, whenever a test run is required, the experiment can simply be executed. Optionally the results of the test run can be saved, either as just a report of the last experiment, or as an annotated data set that includes the predictions so it can be evaluated externally. After inspecting the results, the developer can continue working on the algorithm, for example by adjusting for possible errors that were encountered during the last test run.

3.1. Design Patterns

Adhering to software engineering best practices, we have designed the framework using several existing architecture patterns. The simplified UML class diagram in Figure 2 shows how the main

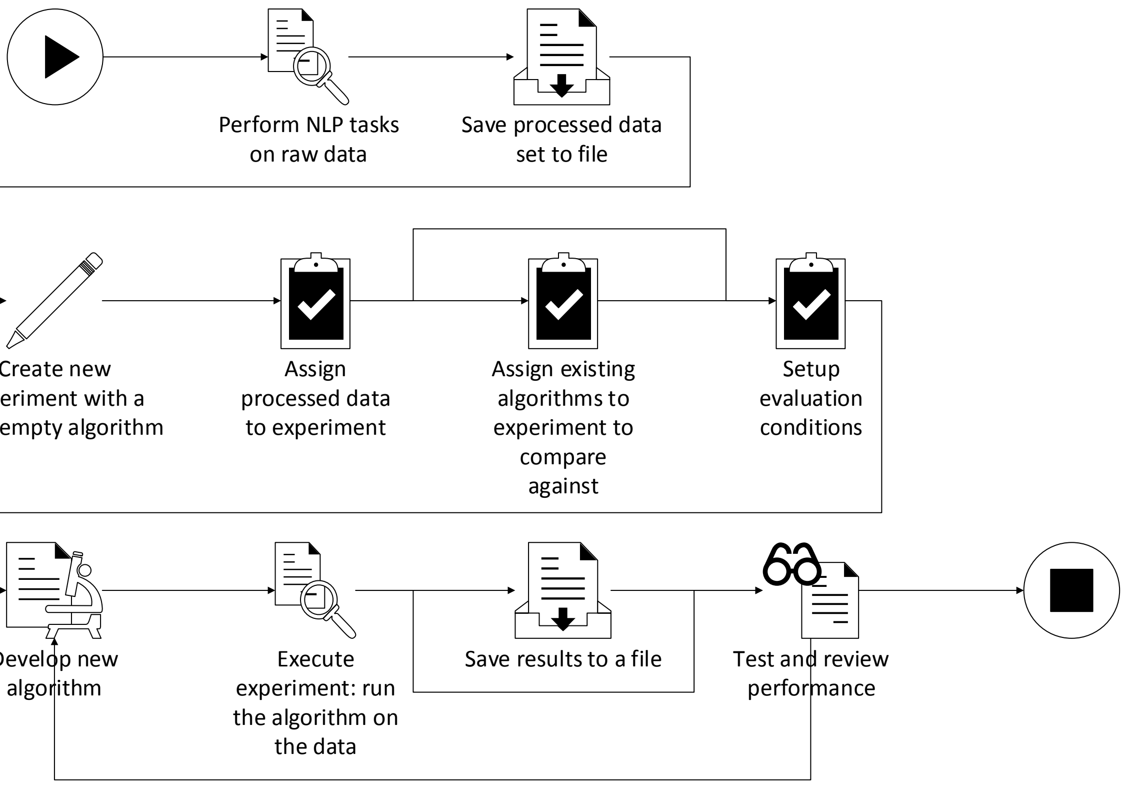

Perform NLP tasks on raw data
Save processed data set to file
Create new experiment with a new, empty algorithm
Assign processed data to experiment
Assign existing algorithms to experiment to compare against
Setup evaluation conditions
Develop new algorithm
Execute experiment: run the algorithm on the data
Save results to a file
Test and review performance

Figure 1: The basic developer workflow within Heracles

components of the framework are organized according to various design patterns. The first major pattern is a Layer pattern (Taylor et al., 2009), separating the various functionalities within the framework. One can distinguish between three layers, each having its own function and within one layer, various implementations can easily be swapped for one another. The first layer is the *data layer*, where datasets are read from various raw sources and processed data models are written back to various file formats. Different implementations of this *data layer* allow for data sets in different formats to be loaded into the Heracles framework. Currently, implementations exist for reading and writing various XML and JSON formats. The second layer is the *natural language processing layer*, where all the required natural language processing is performed so that the developer has access to high level linguistic features when developing a text mining algorithm. Again, implementations of this layer may vary, depending on, for example, the natural language of the dataset. The third layer is the *algorithm layer*, which is the developed algorithm. Naturally, many different algorithms can

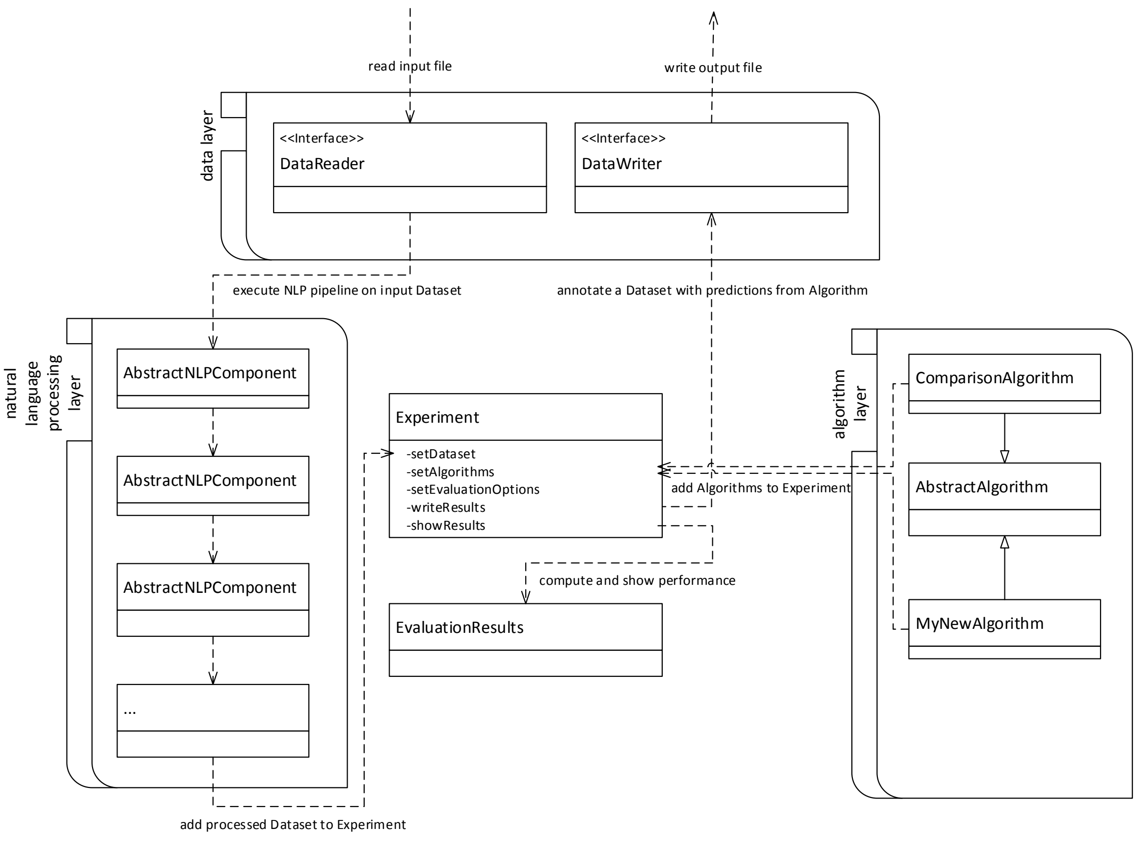


Figure 2: The main components of the framework, organized by layer

be created that will all match the signature of this layer. The first layer is defined as an interface which any implementation of that layer needs to adhere to, to ensure compatibility between the layers. The second and third layer are defined using a Template Method pattern (Gamma et al., 1994) (i.e., an abstract class), which provides not only the interface but also additional functionality that is shared across all implementations of this layer.

The communication between the three main layers is defined using the internal data model, which in turn can be viewed as an API the developers can use in the *algorithm layer* to access data features. In that capacity, the data model, once finished, becomes read only and, to that end, implements the Iterator pattern (Taylor et al., 2009), exposing the data using iterators to prevent access to the underlying data representation. The communication between the layers is as follows. First, the *data layer* reads a raw data file and creates an internal data model that reflects as much of the original information as possible. It then passes the partially constructed data model to the *natural language processing layer*, which will complete the data model and make it read only, before passing it on to the *algorithm layer*, where it will be used to train and test the given algorithm implementation. For example, if the dataset already provides the text split in sentences, these sentence splits will be used when creating the data model. Otherwise, the *natural language processing layer* will execute a component that will try to determine the best places to split the text into proper sentences. Thus, utilizing information provided in the data file is favored over retrieving that same information using natural language processing techniques. Hence, the usually human-annotated information already in the data file is strictly complemented with linguistic information from the *natural language processing layer*. It is up to the `data layer` implementation to properly execute this behavior.

Benefits of using this Layer pattern are that implementations can vary within a given layer. This makes the framework flexible and extendable. Different language-specific implementations for the *natural language processing layer* can easily be swapped in, just like different data readers, so practically any textual data set can be loaded in the framework. For the *algorithm layer*, the main benefit is that any algorithm, developed according to the specifications of the layer, can be automatically executed and tested without any additional effort from the developers side. The downside of using a Layer pattern is that in the rare instance of having to update the communication between the layers, some manual refactoring is needed.

Within the *natural language processing layer* and the *algorithm layer*, a Pipeline pattern (Gamma et al., 1994) can be discerned. All natural language processing components are modular and they can be executed over a given data model. Each module is labeled with the task it performs as well as with a list of prerequisite tasks that have to be performed before this module can be executed. In this way, multiple implementations of the same task can exist within the framework, and depending on the overall goal, a specific module can be selected. When a module has been executed, the data model is labeled with that task, enabling the framework to automatically check whether it is possible (or necessary) to run a certain module on a given data model.

For the *algorithm layer*, the Pipeline pattern is more static, with a general set of steps that is the same for all algorithms: preprocessing of the data, training of the algorithm, predicting for unseen data using the trained algorithm, and evaluating the predicted annotations. The series of steps that together comprise an algorithm's execution is discussed in more detail in a subsequent section. The main benefit of the Pipeline pattern for the *algorithm layer* is that it allows for rapid prototyping. Every algorithm will have predefined methods (i.e., stubs) which will be replaced with the logic of the algorithm. Methods that are not relevant for a given scenario can simply be left as stubs. For example, algorithms that do not require a supervised training phase can leave the `train` method empty. To provide machine learning support in this layer, two machine learning libraries have been integrated into Heracles: Weka (Witten and Frank, 2005), which supports many statistical machine learning methods, and Watchmaker (Watchmaker, 2018), for evolutionary computing support.

To utilize all three layers, a Builder pattern (Bloch, 2008) is used that creates an *experiment* using method chaining. Building up an experiment using method chaining is a convenient way to provide all the necessary information, such as which data set to use, which algorithms to run and compare, how many runs to make, whether to use cross-validation or not, etc. Depending on the options set by the developer, an experiment ends by providing the performance results or by providing an annotated dataset. The latter is useful when evaluation is done externally. Encapsulating experiments in this manner allows the framework to record all run experiments for archival purposes, and since experiments are independent from each other, multiple experiments can be run, both in a serial and parallel manner.

3.2. Class Architecture

The central component in the simplified UML class diagram in Figure 2 is the `Experiment` class. This class has methods to assign a `Dataset` and one or more `Algorithm`s to it. It also has various evaluation options, such as methods to use cross-validation, or methods to repeat the experiment a number of times. The `Experiment` class has two output modes: one is to use an `Algorithm` to annotate a dataset, the other uses an already annotated dataset to test and compare the performance of one or more `Algorithm`s. The latter has built-in support for significance testing, using a paired t-test. The data model, contained in the `Dataset` is passed around and is not shown in this diagram.

A `Dataset` object is created using an implementation of the `DataReader` interface. An object implementing the `DataReader` interface is expected to provide a read method that creates a

`Dataset`, most likely by reading some raw textual data from a (set of) files. It will then count on a series of components in the *NLP layer* to do the necessary natural language processing so that a completely processed `Dataset` can be created. Since some data sets already provide a certain amount of structure or annotations, the `DataReader` is expected to create as much of the `Dataset` as it can, letting the components in the *NLP layer* do the rest. Using the interface design principle, an array of different `DataReader`s can be used to allow data sets in different formats to be processed in the framework. If a `DataReader` does not yet exist for a given data set, it is always possible to build one and plug it in.

The same modular approach is chosen for the NLP layer, where multiple subclasses of the `AbstractNLPComponent` template can be chained to form an NLP pipeline. Each concrete subclass can have its own requirements which are checked before attempting to execute this component. For instance, before being able to perform lemmatization, which is labeling each word with its dictionary form, each word first needs to labeled with its part-of-speech tag, which is the word type (e.g., noun, verb, etc.). Hence, the lemmatization component will check whether the part-of-speech tagging component has already been executed and will throw an error if that is not the case. Thus, the developer can freely choose the components that are going to be used as long as the individual components' requirements are met. Currently, Heracles has wrappers for the majority of the components of the Stanford CoreNLP (Manning et al., 2014) library, a wrapper for the CLULAB Rhetorical Structure Theory parser (Surdeanu et al., 2015), and a component for matching text with concepts in a knowledge base (i.e., ontologies) using the Jena (Apache Jena, 2018) library for ontology communication. A visualization of all the components in the implementation of the *NLP layer* for English is given in Figure 3.

Given a loaded `Dataset`, one can execute an `Algorithm` object on that data, which will result in one or more `EvaluationResults` objects being returned. Since one `Algorithm` can perform multiple tasks that are evaluated separately, it is possible that multiple `EvaluationResults` are generated, one for each task. The `Algorithm` class is an abstract class, providing a template for all new algorithms that are to be developed. The main task of the user is to build their own algorithm and run it in the framework. By following the given template in `Algorithm`, the framework is able

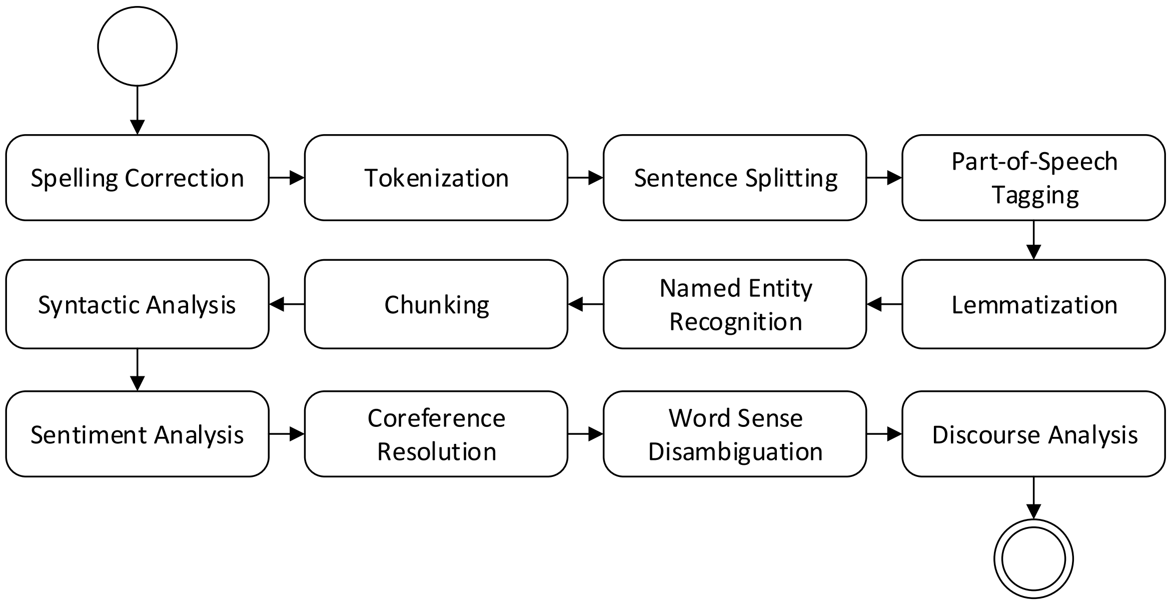


Figure 3: Activity diagram of the complete English NLP layer.

to automate many processes, including giving the right data to the right method (e.g., provide just the training data to the train method) and evaluating the output of the algorithm. The predictions that an algorithm makes are stored locally within the algorithm instance itself, so the loaded data model does not change. In fact, it is read-only to prevent data leakage or data corruption.

3.3. Algorithm Evaluation Procedure

As mentioned before, the framework provides a template to use when developing custom algorithms to ensure every algorithm can be run and evaluated automatically by the framework. The process of running an algorithm is shown as a UML sequence diagram in Figure 4. The first step is to load a `Dataset` using a `DataReader`, which can be one provided with the framework or a custom built one. Then the developed algorithm is instantiated. To distinguish between custom functionality and functionality provided in the template, the `Algorithm` has two distinct lifelines, one for the processes described in the template and one for the processes described in the developed instantiation, or subclass, of the template. Both algorithm and data set are provided to the `Experiment`, which keeps track of those. Then, after setting the necessary evaluation options, the experiment is run.

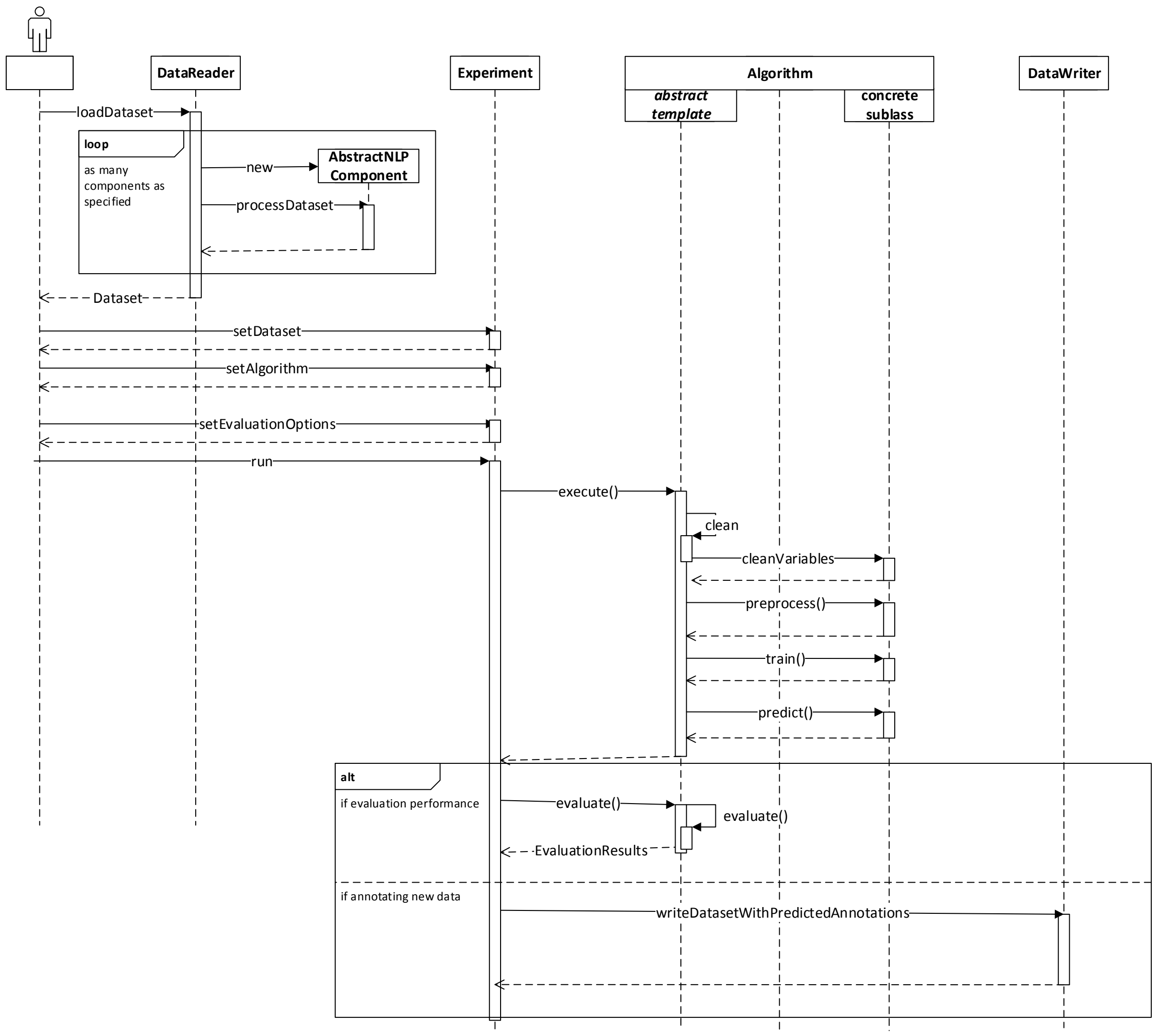


Figure 4: Sequence diagram of running an algorithm in the framework

The experiment splits the dataset in training and test set, depending on the evaluation options, and provides it to the algorithm. The `execute()` method, defined in the template, will go through all the required steps in order to successfully evaluate the performance of this algorithm. The first step is to clean the algorithm data, making sure that there are no results from previous runs leaking into this run. This is of special importance since the predictions of an algorithm are stored locally within the `Algorithm` object so, if not for this cleaning step, predictions would spill over from one run to the next, polluting the evaluation results. One part of the cleaning is to call the `cleanVariables()` method in the developed subclass. The developer is responsible for making sure that any variables that are trained on the data are initialized in this method, so they are reset when a new execution cycle begins. If any parameter of the model used by the developed algorithm is not properly reset, the algorithm could potentially be using information from previous runs which it should have no access to.

The next step is to call the `preprocess(allData)` method. In this method the developer gets access to the whole dataset to do some preprocessing of the data, if necessary. This preprocessing should not use any of the information that this same algorithm tries to predict, since that would defeat the purpose of out-of-sample testing. One can think of activities like counting word frequencies, or performing and caching lookups in knowledge bases. Any information gained in this procedure should be stored locally in the developed algorithm. The next phase is to train the algorithm using the provided training data. Here, the algorithm is supposed to look at the information it needs to predict in order to tweak any parameters, set model weights, etc.

Then, the algorithm is ready to process the test data, which is that part of the data which was excluded from the training data to perform out-of-sample testing. The developed algorithm should predict whatever information it is targeted to predict (i.e., a sentence-level sentiment analysis algorithm will predict a sentiment value for each sentence). These predictions are stored locally in a variable that is defined in the template. Because of that, the `evaluate()` method, which is defined in the template, will have access to these predictions and it will compare those with the provided human annotations in order to compute a performance score for the developed algorithm. For each task the developed algorithm provides predictions for, the evaluation method will create a `EvaluationResults` object to record the performance. All `EvaluationResults` are returned to the user at the end of the process.

Alternatively, the algorithm can use the `predict()` method to annotate data that does not have manually assigned labels. The output will then consist of the automatically annotated data set, and no performance evaluation is performed as the predicted annotations cannot be compared against a gold standard.

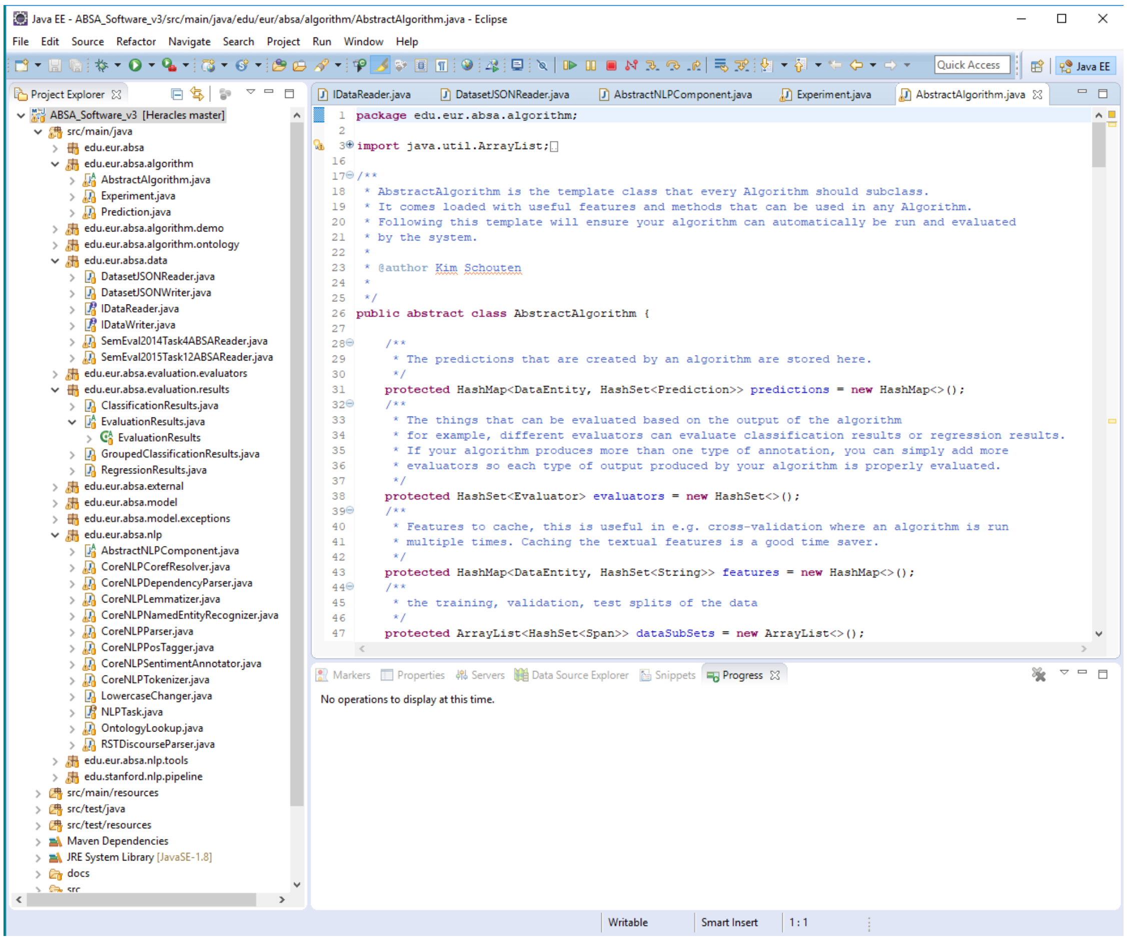
A software snapshot of the proposed framework can be found in Figure 5. The sequence diagram of Figure 4 relates to this snapshot of the proposed framework in the following way.

In the first step, where we load a `Dataset` using a `DataReader`, a public interface `IDataReader` (as well as implementations of this public interface for XML and JSON files) is provided in the package "edu.eur.absa.data". The `AbstractNLPCompontent` class is provided in the "edu.eur.absa.nlp" package, which is used to do the natural language processing. Both the `Experiment` class and `Algorithm` template (`AbstractAlgorithm`) are provided in the package "edu.eur.absa.algorithm". As explained before, in the `evaluate()` method, the `EvaluationResults` objects are created. This class can be found in the package "edu.eur.absa.evaluation.results". In order to write an output file, a public interface `IDataWriter` is provided in the package "edu.eur.absa.data". Also, implementations of `IDataWriter` for writing XML and JSON files are provided in this package.

3.4. Internal Data Model

The data model that is created by the `DataReader` and NLP Pipeline components contains all the linguistic data needed for an algorithm to perform the text mining task at hand. In Figure 6, a UML class diagram is given for the data model as created by the English NLP Pipeline. Note that the data model itself is extendable and that not every component is mandatory. There are three main subclasses of `DataEntity` that are contained in a `Dataset` object: `Words`, `Spans`, and `Relations`. A `Word` object represents a single word in the text, a `Span` represents a sequence of `Words`, and a `Relation` object models an arbitrary, directed relation between two instances of `DataEntity`.

While there can be many types of `Span` objects (e.g., sentences, documents, named entities, etc.), the `Dataset` object knows which type of `Span` is the top-level `Span`. This top-level `Span` is the



```java
package edu.eur.absa.algorithm;

import java.util.ArrayList;[]

/**
 * AbstractAlgorithm is the template class that every Algorithm should subclass.
 * It comes loaded with useful features and methods that can be used in any Algorithm.
 * Following this template will ensure your algorithm can automatically be run and evaluated
 * by the system.
 *
 * @author Kim Schouten
 *
 */
public abstract class AbstractAlgorithm {

    /**
     * The predictions that are created by an algorithm are stored here.
     */
    protected HashMap<DataEntity, HashSet<Prediction>> predictions = new HashMap<>();
    /**
     * The things that can be evaluated based on the output of the algorithm
     * for example, different evaluators can evaluate classification results or regression results.
     * If your algorithm produces more than one type of annotation, you can simply add more
     * evaluators so each type of output produced by your algorithm is properly evaluated.
     */
    protected HashSet<Evaluator> evaluators = new HashSet<>();
    /**
     * Features to cache, this is useful in e.g. cross-validation where an algorithm is run
     * multiple times. Caching the textual features is a good time saver.
     */
    protected HashMap<DataEntity, HashSet<String>> features = new HashMap<>();
    /**
     * the training, validation, test splits of the data
     */
    protected ArrayList<HashSet<Span>> dataSubSets = new ArrayList<>();
```

No operations to display at this time.

Figure 5: Software snapshot of the proposed framework, which relates to the sequence diagram of Figure 4

textual unit, modeling pieces of text that are independent from the other pieces of text that exist in this `Dataset`. For example, a `Dataset` might contain user reviews about a certain product. Then each review is independent from each other review and thus the review is the textual unit. The sentences within each review, however, are semantically related to each other and hence cannot be the top-level `Span` or textual unit.

Since `Spans` are sequences of `Words`, and there can be many types of, potentially overlapping, `Spans`, the framework provides many methods to get the `Spans` of interest, based on type, which textual unit they belong to, and whether or not they overlap or touch a specific `Span`. `Spans` implement the `Iterable<Word>` interface which ensures that they can be used with the `for...each` construct in a natural way.

Relations can be used to create relations between Words, such as grammatical dependencies, or to create relations between Spans, such as discourse relations between parts of the text. Furthermore, Relations can be used to model a relation between a Word and a Span, for example to denote the head word of a parse tree constituent.

This data model is flexible enough to model any natural language phenomenon without the need for additional classes. This makes serialization a practical endeavor, as the provided readers and writers for the data model will cover any use of this data model without the need for programmatic modification.

Next to the software snapshot provided in Figure 5, another software snapshot is provided in Figure 7. From this figure, we see that all classes from the class diagram in Figure 6 are provided in the package "edu.eur.absa.model".

4. Sentiment Analysis Use Case

The framework we propose has already been used in several concrete applications. The most prominent use case has been the development of new sentiment analysis algorithms. Sentiment analysis is a popular research area (Pang and Lee, 2008; Liu, 2012; Feldman, 2013), both in academia and in business, because it is a challenging problem with many useful business applications. The central problem is to find the sentiment expressed in a certain text, with varying degrees of granularity. Sentiment can be determined for complete documents or for full sentences, but even more interesting is the task of aspect-level sentiment analysis (Schouten and Frasincar, 2016), where sentiment is computed with respect to the entities and aspects of entities that are actually discussed within the text. This task naturally breaks down into aspect detection and sentiment analysis for aspects. Aspects can be explicitly mentioned in a sentence, which means that there is a sequence of words (e.g., "glass of wine") that denotes that aspect, but they can also be implied by the sentence as a whole (e.g., "they threw the dishes on our table!", implying the service aspect). Each aspect has its own associated sentiment score that has to be determined.

One of the more recent projects that has been developed using Heracles is an ontology-enhanced algorithm (Schouten and Frasincar, 2018) for aspect-based sentiment analysis. In this work, an

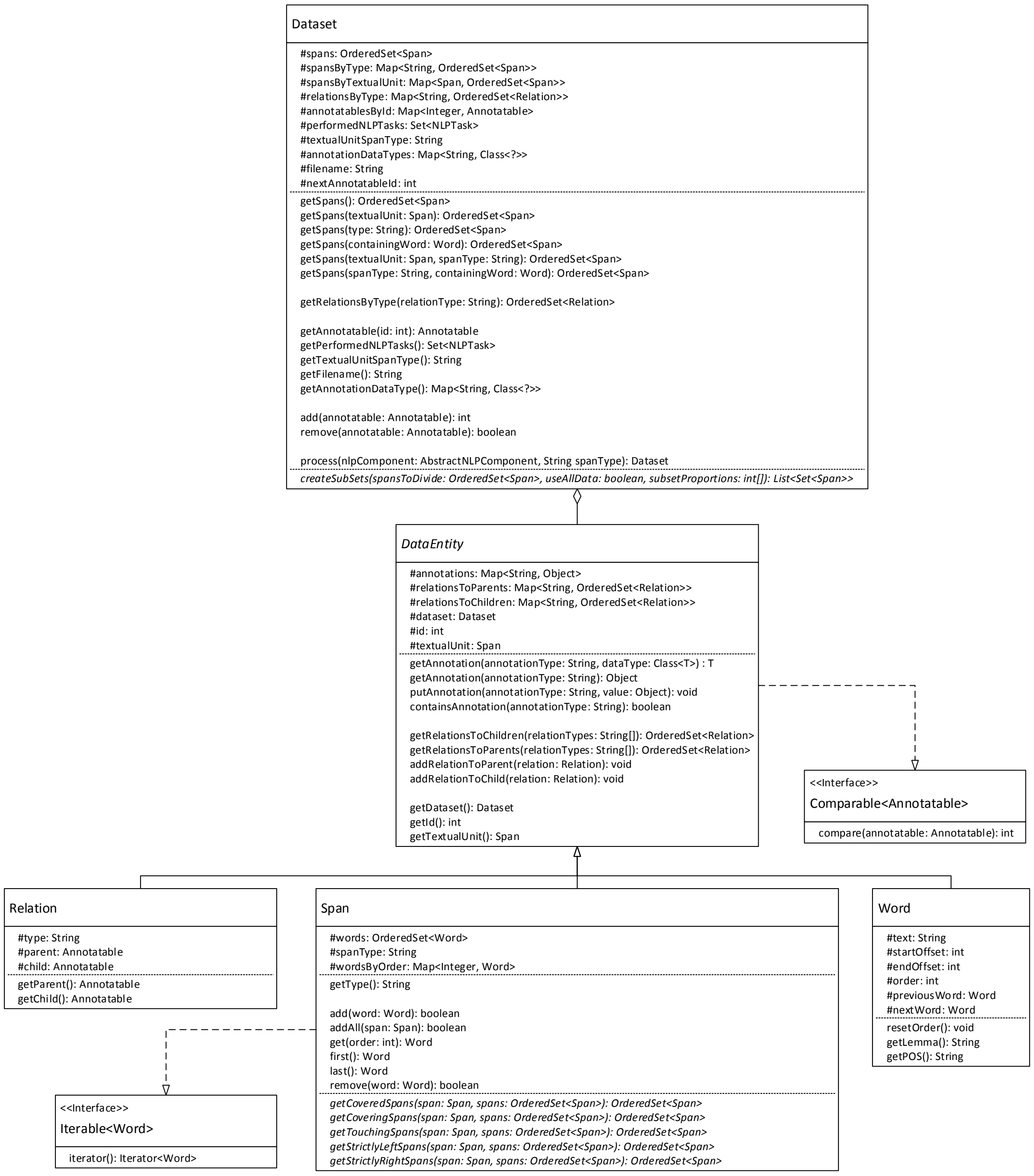
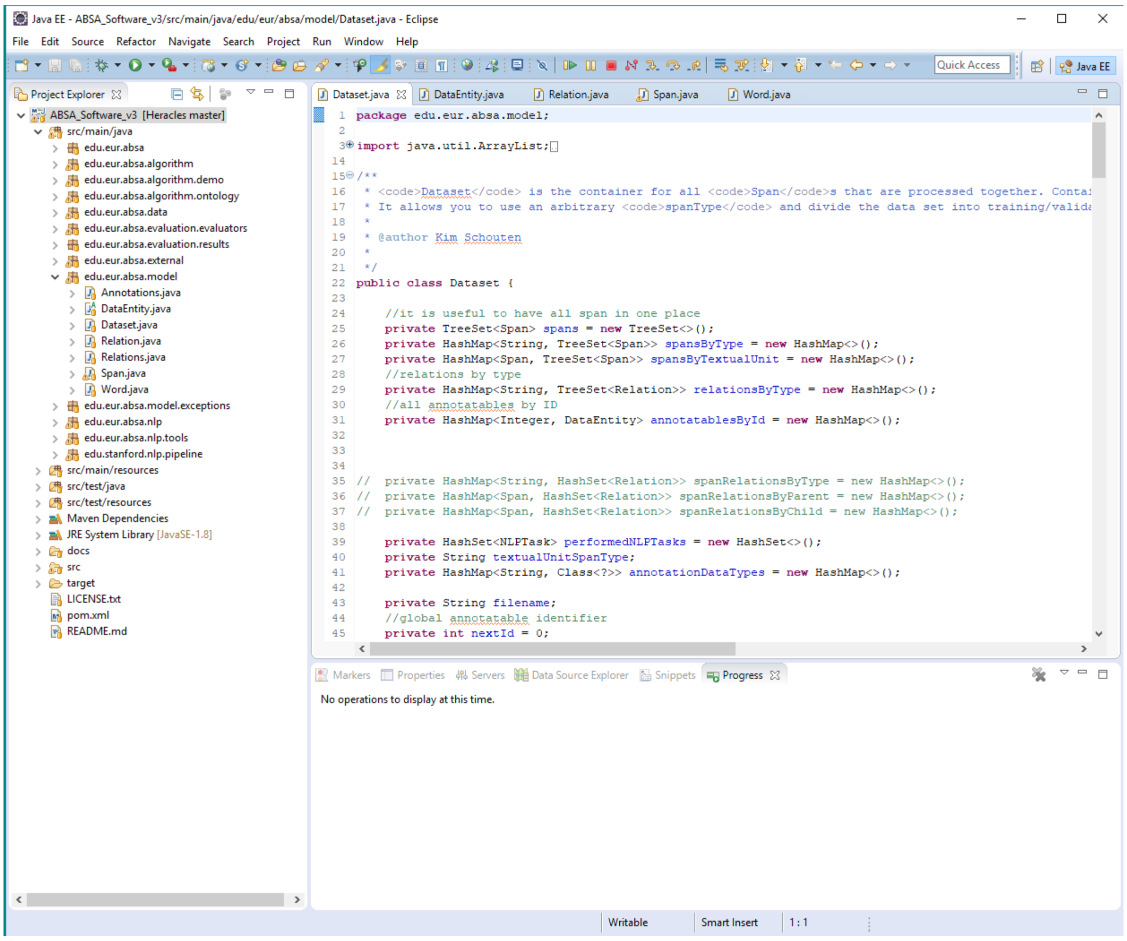

Dataset
#spans: OrderedSet<Span>
#spansByType: Map<String, OrderedSet<Span>>
#spansByTextualUnit: Map<Span, OrderedSet<Span>>
#relationsByType: Map<String, OrderedSet<Relation>>
#annotatablesById: Map<Integer, Annotatable>
#performedNLPTasks: Set<NLPTask>
#textualUnitSpanType: String
#annotationDataTypes: Map<String, Class<?>>
#filename: String
#nextAnnotatableId: int
getSpans(): OrderedSet<Span>
getSpans(textualUnit: Span): OrderedSet<Span>
getSpans(type: String): OrderedSet<Span>
getSpans(containingWord: Word): OrderedSet<Span>
getSpans(textualUnit: Span, spanType: String): OrderedSet<Span>
getSpans(spanType: String, containingWord: Word): OrderedSet<Span>
getRelationsByType(relationType: String): OrderedSet<Relation>
getAnnotatable(id: int): Annotatable
getPerformedNLPTasks(): Set<NLPTask>
getTextualUnitSpanType(): String
getFilename(): String
getAnnotationDataType(): Map<String, Class<?>>
add(annotatable: Annotatable): int
remove(annotatable: Annotatable): boolean
process(nlpComponent: AbstractNLPComponent, String spanType): Dataset
createSubSets(spansToDivide: OrderedSet<Span>, useAllData: boolean, subsetProportions: int[]): List<Set<Span>>
DataEntity
#annotations: Map<String, Object>
#relationsToParents: Map<String, OrderedSet<Relation>>
#relationsToChildren: Map<String, OrderedSet<Relation>>
#dataset: Dataset
#id: int
#textualUnit: Span
getAnnotation(annotationType: String, dataType: Class<T>) : T
getAnnotation(annotationType: String): Object
putAnnotation(annotationType: String, value: Object): void
containsAnnotation(annotationType: String): boolean
getRelationsToChildren(relationTypes: String[]): OrderedSet<Relation>
getRelationsToParents(relationTypes: String[]): OrderedSet<Relation>
addRelationToParent(relation: Relation): void
addRelationToChild(relation: Relation): void
getDataset(): Dataset
getId(): int
getTextualUnit(): Span
<<Interface>>
Comparable<Annotatable>
compare(annotatable: Annotatable): int
Relation
#type: String
#parent: Annotatable
#child: Annotatable
getParent(): Annotatable
getChild(): Annotatable
Span
#words: OrderedSet<Word>
#spanType: String
#wordsByOrder: Map<Integer, Word>
getType(): String
add(word: Word): boolean
addAll(span: Span): boolean
get(order: int): Word
first(): Word
last(): Word
remove(word: Word): boolean
getCoveredSpans(span: Span, spans: OrderedSet<Span>): OrderedSet<Span>
getCoveringSpans(span: Span, spans: OrderedSet<Span>): OrderedSet<Span>
getTouchingSpans(span: Span, spans: OrderedSet<Span>): OrderedSet<Span>
getStrictlyLeftSpans(span: Span, spans: OrderedSet<Span>): OrderedSet<Span>
getStrictlyRightSpans(span: Span, spans: OrderedSet<Span>): OrderedSet<Span>
Word
#text: String
#startOffset: int
#endOffset: int
#order: int
#previousWord: Word
#nextWord: Word
resetOrder(): void
getLemma(): String
getPOS(): String
<<Interface>>
Iterable<Word>
iterator(): Iterator<Word>

Figure 6: Class diagram of the data model used within the framework.



Java EE - ABSA_Software_v3/src/main/java/edu/eur/absa/model/Dataset.java - Eclipse

```
ABSA_Software_v3 [Heracles master]
  src/main/java
    edu.eur.absa
    edu.eur.absa.algorithm
    edu.eur.absa.algorithm.demo
    edu.eur.absa.algorithm.ontology
    edu.eur.absa.data
    edu.eur.absa.evaluation.evaluators
    edu.eur.absa.evaluation.results
    edu.eur.absa.external
    edu.eur.absa.model
      Annotations.java
      DataEntity.java
      Dataset.java
      Relation.java
      Relations.java
      Span.java
      Word.java
    edu.eur.absa.model.exceptions
    edu.eur.absa.nlp
    edu.eur.absa.nlp.tools
    edu.stanford.nlp.pipeline
  src/main/resources
  src/test/java
  src/test/resources
  Maven Dependencies
  JRE System Library [JavaSE-1.8]
  docs
  src
  target
  LICENSE.txt
  pom.xml
  README.md
```

```java
package edu.eur.absa.model;

import java.util.ArrayList;

/**
 * <code>Dataset</code> is the container for all <code>Span</code>s that are processed together. Contai
 * It allows you to use an arbitrary <code>spanType</code> and divide the data set into training/valida
 *
 * @author Kim Schouten
 *
 */
public class Dataset {

    //it is useful to have all span in one place
    private TreeSet<Span> spans = new TreeSet<>();
    private HashMap<String, TreeSet<Span>> spansByType = new HashMap<>();
    private HashMap<Span, TreeSet<Span>> spansByTextualUnit = new HashMap<>();
    //relations by type
    private HashMap<String, TreeSet<Relation>> relationsByType = new HashMap<>();
    //all annotatables by ID
    private HashMap<Integer, DataEntity> annotatablesById = new HashMap<>();



//  private HashMap<String, HashSet<Relation>> spanRelationsByType = new HashMap<>();
//  private HashMap<Span, HashSet<Relation>> spanRelationsByParent = new HashMap<>();
//  private HashMap<Span, HashSet<Relation>> spanRelationsByChild = new HashMap<>();

    private HashSet<NLPTask> performedNLPTasks = new HashSet<>();
    private String textualUnitSpanType;
    private HashMap<String, Class<?>> annotationDataTypes = new HashMap<>();

    private String filename;
    //global annotatable identifier
    private int nextId = 0;
```

No operations to display at this time.

Figure 7: Software snapshot of the proposed framework, which relates to the class diagram of Figure 6

ontology, which is a formally specified knowledge base that holds domain-specific information, is used to improve the performance of a classical machine learning approach in both sub-tasks. To that end, the framework is enriched with an ontology component that uses the Jena (Apache Jena, 2018) library to connect with an ontology and this component provides functionality to search an ontology, add concepts, and get inference results from it. This component is used in a custom-built pipeline module, which is in the NLP layer, that labels the text with all the matching concepts found in the ontology. The ontology itself has concepts that cover aspects as well as sentiment.

In this paper (Schouten and Frasincar, 2018), the goal is to extract the sentiment of content creators, such as the writers of consumer reviews, and to aggregate this information into easy to digest overviews, infographics, or dashboards. Since reviews often go into detail about certain characteristics of the entity under review, it is useful to go one step further and perform aspect-based sentiment analysis.

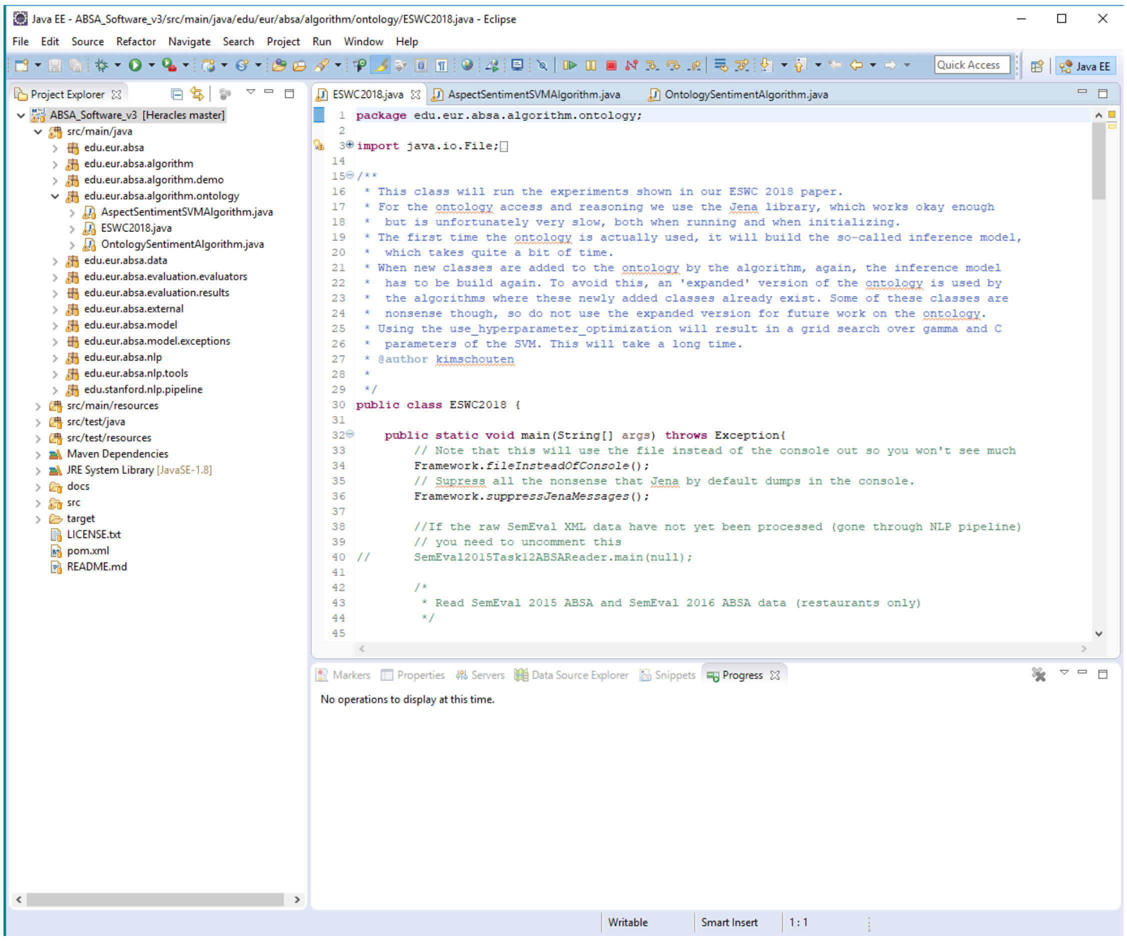
The previous work focuses only on sentiment analysis, which is achieved by clearly distinguishing between three types of sentiment words: generic sentiment words that always have the same sentiment value regardless of the context, aspect-specific sentiment words that infer the presence of a single aspect and are only applicable to that aspect (e.g., "rude" for the service aspect), and context-dependent sentiment words that are applicable to more than one aspect, but not necessarily all of them, and that may have different sentiment values for different aspects (e.g., small being generally negative for portions, but usually positive for price). For this sentiment analysis, a two-stage approach is used. In the primary stage, an ontology is used to find and infer sentiment for the current aspect, and, if successful, that becomes the prediction of the method. Only when the ontology either finds both positive and negative signals, or none at all, we employ a Support Vector Machine (SVM) model to predict the sentiment. This secondary, or backup, model is a slightly improved Bag-of-Words model that does not use ontology features. With just a small number of changes, the SVM model could be replaced by a different model, such as a Random Forest, or a Logistic Regression.

A snapshot of the software for the paper on ontology-driven sentiment analysis of product and service aspects is provided in Figure 8. From this figure, it becomes clear that three classes had to be written: one class to implement the SVM model (`AspectSentimentSVMAlgorithm`), one class to implement the ontology-driven model (`OntologySentimentAlgorithm`), and one class to run the experiments (`ESWC2018`), using the other two classes. In order to get this use case running, the following steps have to be taken:

- Open Eclipse, import a project with Git, choose "Clone URI", and use the URI from the Heracles repository;
- Find the `ESWC2018` class, which is provided in the package "edu.eur.absa.algorithm.ontology";
- Uncomment line 40 when running this class for the first time, such that the SemEval files go through the NLP pipeline;

- Lines 87-92 have a list of experiments that were run for the paper. Comment or uncomment these to choose which one(s) to run;
- Run the ESWC2018 class (it has its own main function).

Note: you first have to create an output directory in the Heracles repository, in order to have the console output to be written to a text file with the `fileInsteadOfConsole()` function.


Java EE - ABSA_Software_v3/src/main/java/edu/eur/absa/algorithm/ontology/ESWC2018.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Project Explorer
ABSA_Software_v3 [Heracles master]
src/main/java
edu.eur.absa
edu.eur.absa.algorithm
edu.eur.absa.algorithm.demo
edu.eur.absa.algorithm.ontology
AspectSentimentSVMAlgorithm.java
ESWC2018.java
OntologySentimentAlgorithm.java
edu.eur.absa.data
edu.eur.absa.evaluation.evaluators
edu.eur.absa.evaluation.results
edu.eur.absa.external
edu.eur.absa.model
edu.eur.absa.model.exceptions
edu.eur.absa.nlp
edu.eur.absa.nlp.tools
edu.stanford.nlp.pipeline
src/main/resources
src/test/java
src/test/resources
Maven Dependencies
JRE System Library [JavaSE-1.8]
docs
src
target
LICENSE.txt
pom.xml
README.md
ESWC2018.java
AspectSentimentSVMAlgorithm.java
OntologySentimentAlgorithm.java
Markers
Properties
Servers
Data Source Explorer
Snippets
Progress
Writable
Smart Insert
1 : 1

```java
package edu.eur.absa.algorithm.ontology;

import java.io.File;

/**
 * This class will run the experiments shown in our ESWC 2018 paper.
 * For the ontology access and reasoning we use the Jena library, which works okay enough
 *  but is unfortunately very slow, both when running and when initializing.
 * The first time the ontology is actually used, it will build the so-called inference model,
 *  which takes quite a bit of time.
 * When new classes are added to the ontology by the algorithm, again, the inference model
 *  has to be build again. To avoid this, an 'expanded' version of the ontology is used by
 *  the algorithms where these newly added classes already exist. Some of these classes are
 *  nonsense though, so do not use the expanded version for future work on the ontology.
 * Using the use_hyperparameter_optimization will result in a grid search over gamma and C
 *  parameters of the SVM. This will take a long time.
 * @author kimschouten
 *
 */
public class ESWC2018 {

    public static void main(String[] args) throws Exception{
        // Note that this will use the file instead of the console out so you won't see much
        Framework.fileInsteadOfConsole();
        // Supress all the nonsense that Jena by default dumps in the console.
        Framework.suppressJenaMessages();

        //If the raw SemEval XML data have not yet been processed (gone through NLP pipeline)
        // you need to uncomment this
//      SemEval2015Task12ABSAReader.main(null);

        /*
         * Read SemEval 2015 ABSA and SemEval 2016 ABSA data (restaurants only)
         */

```

No operations to display at this time.

Figure 8: Software snapshot for the paper on ontology-driven sentiment analysis of product and service aspects (Schouten and Frasincar, 2018)

The paper makes use of restaurant reviews from SemEval 2015 and SemEval 2016, which are datasets which consist of reviews with sentiment-labeled aspects. The results in the paper show that the proposed method has a highly competitive performance of over 80% accuracy on both SemEval 2015 and SemEval 2016 data, significantly outperforming the considered baselines. As a matter of fact, the proposed method has the highest accuracy compared to the competing methods on the SemEval 2015 data, and the third highest accuracy compared to the competing methods on the SemEval 2016 data.

The machine learning component comes from the linked Weka (Witten and Frank, 2005) library, and this means that the various input features have to be specified in terms of Weka objects. Input features are things like which words are present in a sentence, which ontology concepts are found in the sentence, etc. While it can be considered an extra burden, having to transform between the data model used in Heracles, and the data model used by Weka, there are many advantages. The biggest one is that once transformed, there is no cost in transitioning to a different machine learning model. Furthermore, Weka has built-in options for feature selection and meta-parameter optimization (like the c and γ for an SVM). By taking the effort to link to Weka and transform to its data model, all this functionality comes at almost no extra costs in terms of development time.

We have evaluated the proposed framework by setting up an SVM algorithm for aspect-based sentiment analysis using a Bag-of-Words approach, in both Heracles and in R. To predict the sentiment of an aspect, all the words in the sentence the aspect is in are used, plus the aspect category. The aspect category is thus the only differentiator for multiple aspects appearing in the same sentence. The dataset used for this task is that of SemEval 2016, and the performance measures are obtained by training on the official training data and evaluating on the official test data. Using Heracles, it took about 1.5 hours to setup the SVM algorithm, with a performance of 77% accuracy. In R it took a bit over 2 hours to setup the SVM algorithm, with a performance of 65% accuracy. The setup in Heracles was slowed down by the generic data model, which makes it relatively hard to use for easy algorithms such as the one employed here. In R, it took some time to figure out the format that the R package for SVM wants the data to be in. Extra time was also needed in R for programming things like computing performance, which was not needed in Heracles.

Over the last three years, multiple papers on sentiment mining have been published based on research using the Heracles software framework. These papers deal with various sentiment mining topics: review-level aspect-based sentiment analysis (de Kok et al., 2018a,b), ontology-enhanced aspect-based sentiment analysis (Schouten and Frasincar, 2018; de Heij et al., 2017; Schouten et al., 2017b), feature extraction and selection for aspect-based sentiment analysis (Schouten et al., 2016a,b), implicit aspect and/or category detection (Schouten et al., 2018; Dosoula et al., 2016a,b), and Rhetorical Structure Theory-driven aspect-based sentiment analysis (Hoogervorst et al., 2016). The diversity of the research topics supported by Heracles in these papers shows the versatility of the proposed framework.

Also, during the development of Heracles, it has been used by several groups of students for assignments and seminar projects, as well as individual students for thesis projects. Over the last four years we have gathered feedback from around 10 teams of roughly 4 students each, and from about 10 individual students. The individual students were oftentimes also involved in one of the groups, but as they have additional experience with the system, their input has been particularly valuable. Most students are in their senior year of the econometrics bachelor but some are doing an econometrics master program. Both programs are part of the Erasmus School of Economics at the Erasmus University Rotterdam. Asking the students directly for feedback has yielded useful information which has lead to further improvements to the system's architecture and implementation.

Next to this received qualitative feedback, we have gathered more quantitative feedback by conducting a questionnaire among the same group of students who used Heracles. The questionnaire was answered by 13 students. A summary of the results obtained from the questionnaire is provided in the next two subsections.

4.1. Strengths

Students appreciated the framework as a starting point, allowing them to use many functions that were already provided. In particular, the methods for reading and writing data sets, running data through a natural language processing pipeline, and the automatic evaluation where regarded as a major time saver. This is similar to other frameworks who provide basic functionality out-of-the-box.

Furthermore, the data model was considered to be easy to work with, mostly due to the fact that there are only a handful of classes involved. Because the Heracles framework has a generic data model that works with many text analysis tasks instead of having a collection of specific classes for each supported task, new users do not get lost in myriad of classes and options. This seems to be verified in the questionnaire, where we asked the students to rate the organisation of the Heracles framework. The results of this question are displayed in Figure 9, from which we observe that 69.2% of the students rated the organisation with a score of 4 or 5 out of 5.

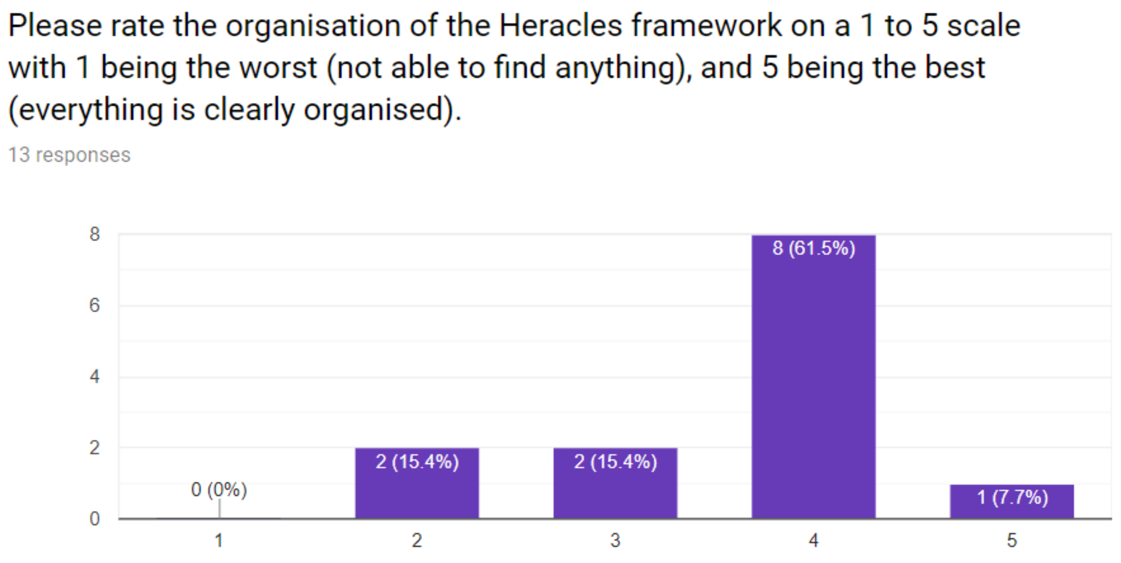


Figure 9: Questionnaire results on the organisation of the Heracles framework

While developing with any API or existing framework requires a significant time investment in order to get acquainted with the ins and outs of the code, the Heracles framework does not have a very steep learning curve. It took individual bachelor students, who did only an introductory Java course, only about two weeks to get up to speed. The code adheres to the standard object-oriented programming paradigm, which is what new Java users would know. In the questionnaire, we asked the students to rate the ease-of-use (or learning curve) of the Heracles framework. Figure 10 shows the results of this question, where 30.8% of the students rated ease-of-use with a score of 4 or 5 out of 5, and 53.8% of the students gave a score of 3.

Another positive aspect, as noted by many students, is the fact that many example implementations are provided that show how to perform certain tasks. For example, the framework comes with a number of algorithms already implemented so users can see how various concepts are

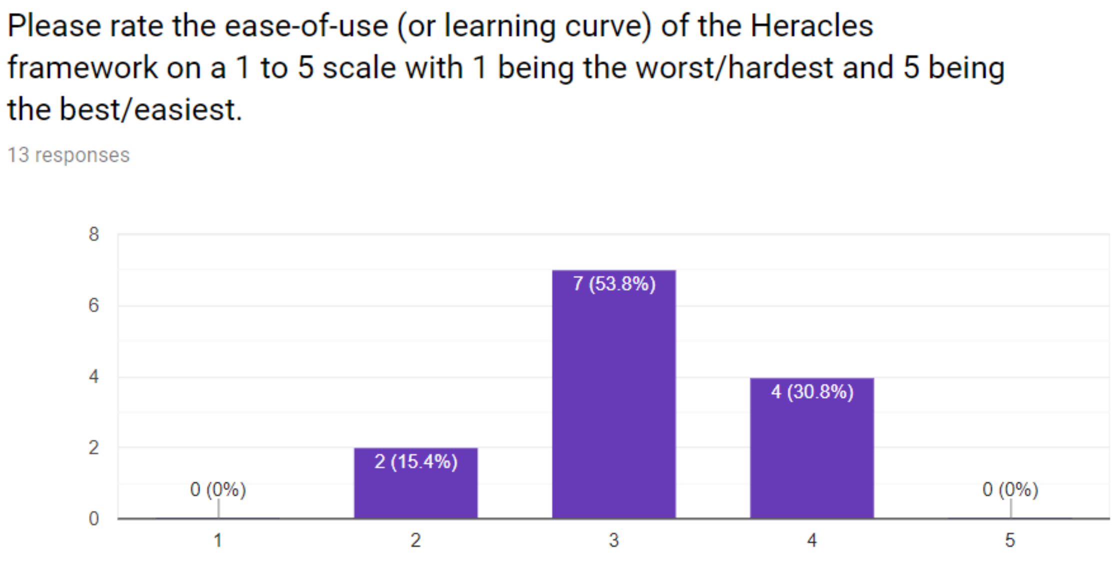


Figure 10: Questionnaire results on the ease-of-use of the Heracles framework

realized. This ranges from very simple classes that show how to work with the `Algorithm` class to published algorithms that are more complex.

Last, we asked the students in the questionnaire whether they would recommend Heracles to someone developing text mining algorithms. As can be seen from Figure 11, 69.2% of the students answered 'Yes' to this question.

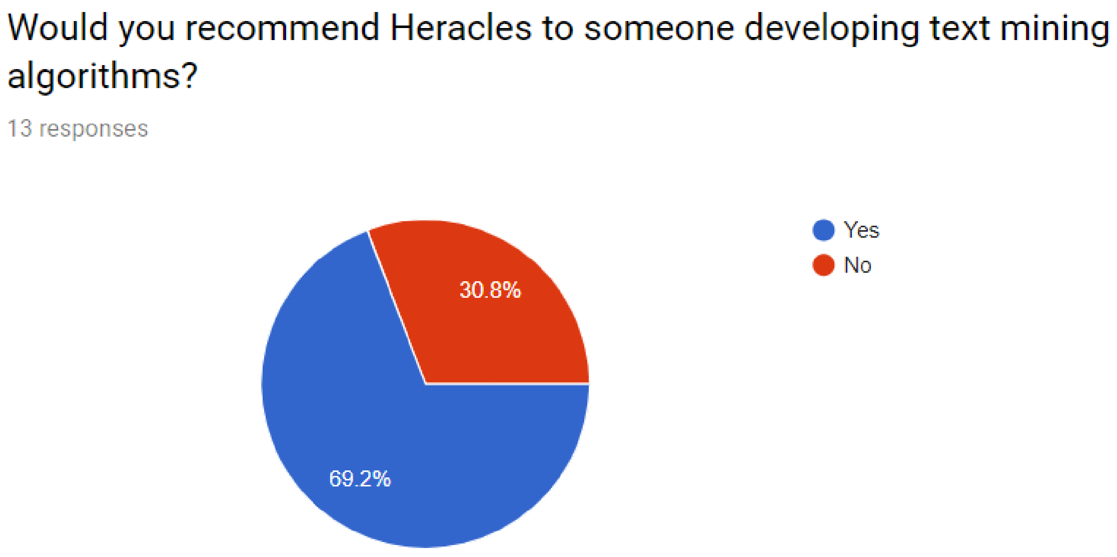


Figure 11: Questionnaire results on recommendation of the Heracles framework

4.2. Weaknesses

Over the years, students have remarked that certain design choices were not always clear. This resulted in not being able to find a specific piece of code in the most logical place. Subsequent iterations have addressed this issue, and this comment was not heard with respect to the latest version.

Another change compared to previous versions is the fact that while previous versions where specifically tailored to sentiment analysis, the current version is not limited to a specific data set or task and can handle a wide variety of text mining tasks (cf. Schouten et al. (2017a) for regression type problems or Boon et al. (2016) for a word sense disambiguation problem). This comes, however, at the cost of a higher level of abstraction, which makes the code harder to understand. The example classes have proven to be really necessary to communicate how the framework should be used.

While the framework is designed to ensure scientifically valid results, it is always possible to change the code of the framework and break this functionality. Currently, development is done within the Heracles software project, so any piece of code can be changed. Putting the Heracles core in a library format, which is thus no longer easily changed, might mitigate this potential problem. While protecting future developers against themselves can be a good thing, it also limits the possible contributions to the framework itself. Such a trade-off would need to be carefully considered, and the exact choice likely depends on the exact scenario in which Heracles is used. Our advice is to keep everything open and accessible, and provide clear instructions so users do not inadvertently change important code. With the various student projects, we did not encounter difficulties with this approach.

5. Conclusions and Future Work

In this paper we have proposed a versatile framework for developing and evaluating text mining algorithms. With the importance of text mining in various business domains (e.g., Fan et al. (2006); Netzer et al. (2012); Kumar and Ravi (2016)), our framework, which is open source, allows for easy experimentation with existing and future text mining algorithms.

In conclusion, we can state that the framework fulfills its objectives in aiding the development process and providing a solid scientific validation procedure for text mining algorithms. The framework is designed to be flexible and extendable, and it has shown to possess these qualities by virtue of the different applications it has already been used for. Hence, we believe that anyone who develops algorithms in the field of text mining, be it in a business or in academia, can benefit from the framework we propose. Users can opt to use the Java implementation we have built ourselves, or they can choose to use the design principles laid out in this work to build or improve their own text mining framework.

As with any software, there are multiple possibilities for improvement. One of the things that would help new users is an active community of users where developers can ask questions and get help. This aspect of software development is an important consideration when deciding which framework or language to use. In terms of the software itself, the most desirable feature that is not yet implemented is support for parallel computing. Since natural language processing and text classification can take quite a long time, providing tools to speed up this process are most welcome. This could range from relatively simple multi-threading to support for cluster computing paradigms like Spark (Zaharia et al., 2016). We also plan to extend the ontology-based solution for sentiment mining (the presented use case) using word embeddings (as the ones produced by word2vec (Mikolov et al., 2013)) in order to be able to map previously unseen concepts to the ontology ones. Such a solution aims to improve the recall of the considered task.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Technical Report. Google. URL: `http://tensorflow.org/`. `download.tensorflow.org/paper/whitepaper2015.pdf`.

Abeel, T., Van de Peer, Y., Saeys, Y., 2009. Java-ML: A Machine Learning Library. Journal of Machine Learning Research 10, 931–934.

Aggarwal, C.C., Zhai, C., 2012. Mining Text Data. Springer Science & Business Media.

Alias-i, 2018. Lingpipe 4.1.2. [WWW document] `http://alias-i.com/lingpipe` (accessed 19th July 2018).

Apache Jena, 2018. Apache Jena. [WWW document] `https://jena.apache.org/` (accessed 19th July 2018).

Apache OpenNLP, 2018. OpenNLP: A Java-based NLP Toolkit. [WWW document] `opennlp.apache.org` (accessed 19th July 2018).

Bethard, S., Carpuat, M., Cer, D., Jurgens, D., Nakov, P., Zesch, T. (Eds.), 2016. Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016). Association for Computational Linguistics.

Biemann, C., Mehler, A., 2014. Text Mining: From Ontology Learning to Automated Text Processing Applications. Springer.

Bird, S., Loper, E., Klein, E., 2009. Natural Language Processing with Python. OReilly Media Inc.

Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent Dirichlet Allocation. Journal of Machine Learning Research 3, 993–1022.

Bloch, J., 2008. Effective Java, 2nd Edition. Addison-Wesley Professional.

Bloomberg, 2018. Finding novel ways to trade on sentiment data. [WWW document] `https://www.bloomberg.com/professional/blog/finding-novel-ways-trade-sentiment-data/` (accessed 30th August 2018).

Boon, F., de Graaf, L., van Ham, D., Kuilboer, H., Wang, Z., 2016. Semantics-Driven Aspect Based Sentiment Analysis. [WWW document] `http://www.kimschouten.com/papers/unpublished/seminar2016_BAQM_ABSASentiment.pdf` (accessed 19th July 2018).

Chan, S.W., Chong, M.W., 2017. Sentiment Analysis in Financial Texts. Decision Support Systems 94, 53 – 64.

Chang, C.C., Lin, C.J., 2011. LIBSVM: A Library for Support Vector Machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M.A., Saggion, H., Petrak, J., Li, Y., Peters, W., 2011. Text Processing with GATE (Version 6). GATE.

Dosoula, N., Griep, R., den Ridder, R., Slangen, R., van Luijk, R., Schouten, K., Frasincar, F., 2016a. Sentiment Analysis of Multiple Implicit Features per Sentence in Consumer Review Data, in: Databases and Information Systems IX, IOS Press. pp. 241–254.

Dosoula, N., Griep, R., den Ridder, R., Slangen, R., Schouten, K., Frasincar, F., 2016b. Detection of Multiple Implicit Features per Sentence in Consumer Review Data, in: Proceedings of the 12th International Baltic Conference on Databases and Information Systems (DB&IS 2016), Springer. pp. 289–303.

Fan, W., Wallace, L., Rich, S., Zhang, Z., 2006. Tapping the Power of Text Mining. Communications of the ACM 49, 76–82.

Farhadloo, M., Patterson, R.A., Rolland, E., 2016. Modeling Customer Satisfaction from Unstructured Data using a BayesianApproach. Decision Support Systems 90, 1 – 11.

Fdez-Riverola, F., Glez-Pea, D., Lpez-Fernndez, H., Reboiro-Jato, M., Mndez, J., 2012. A JAVA Application Framework for Scientific Software Development. Software: Practice and Experience 42, 1015–1036.

Feldman, R., 2013. Techniques and Applications for Sentiment Analysis. Communications of the ACM 56, 82–89.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

de Heij, D., Troyanovsky, A., Yang, C., Scharff, M.Z., Schouten, K., Frasincar, F., 2017. An Ontology-Enhanced Hybrid Approach to Aspect-Based Sentiment Analysis, in: Proceedings of the 18th International Conference on Web Information Systems Engineering (WISE 2017), Springer. pp. 338–345.

Hoogervorst, R., Essink, E., Jansen, W., van den Helder, M., Schouten, K., Frasincar, F., Taboada, M., 2016. Aspect-Based Sentiment Analysis on the Web Using Rhetorical Structure Theory, in: Proceedings of the 16th International Conference on Web Engineering (ICWE 2016), pp. 317–334.

Kilgariff, A., Palmer, M. (Eds.), 1998. Proceedings of the Pilot SensEval. Association for Computational Linguistics.

de Kok, S., Punt, L., van den Puttelaar, R., Ranta, K., Schouten, K., Frasincar, F., 2018a. Review-Aggregated Aspect-Based Sentiment Analysis with Ontology Features. Progress in AI 7, 295–306.

de Kok, S., Punt, L., van den Puttelaar, R., Ranta, K., Schouten, K., Frasincar, F., 2018b. Review-Level Aspect-Based Sentiment Analysis Using an Ontology, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC 2018), ACM. pp. 315–322.

Krekel, H., 2018. execnet: Distributed Python Deployment and Communication. [WWW document] `http://codespeak.net/execnet/` (accessed 19th July 2018).

Kumar, B.S., Ravi, V., 2016. A Survey of the Applications of Text Mining in Financial Domain. Knowledge-Based Systems 114, 128–147.

Lafferty, J.D., McCallum, A., Pereira, F.C.N., 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, in: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Morgan Kaufmann Publishers Inc.. pp. 282–289.

Liu, B., 2012. Sentiment Analysis and Opinion Mining. volume 16 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool.

Lourenço, A., Carreira, R., Carneiro, S., Maia, P., Glez-Peña, D., Fdez-Riverola, F., Ferreira, E.C., Rocha, I., Rocha, M., 2009. @Note: A Workbench for Biomedical Text Mining. Journal of Biomedical Informatics 42, 710 – 720.

Lyubimov, D., Palumbo, A., 2016. Apache Mahout: Beyond MapReduce. CreateSpace Independent Publishing Platform.

Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D., 2014. The Stanford CoreNLP Natural Language Processing Toolkit, in: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Association for Computational Linguistics. pp. 55–60.

McCallum, A.K., 2002. MALLET: A Machine Learning for Language Toolkit. `http://mallet.cs.umass.edu`.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M.J., Zadeh, R., Zaharia, M., Talwalkar, A., 2016. MLib: Machine Learning in Apache Spark. Journal of Machine Learning Research 17, 1235–1241.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed Representations of Words and Phrases and their Compositionality, in: Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS 2013), NIPS. pp. 3111–3119.

Netzer, O., Feldman, R., Goldenberg, J., Fresko, M., 2012. Mine Your Own Business: Market-Structure Surveillance through Text Mining. Marketing Science 31, 521–543.

Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable Parallel Programming with CUDA. Queue 6, 40–53.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., Marsi, E., 2007. MaltParser: A Language-independent System for Data-driven Dependency Parsing. Natural Language Engineering 13, 95135.

Pang, B., Lee, L., 2008. Opinion Mining and Sentiment Analysis. Foundations and Trends in Information Retrieval 2, 1–135.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825–2830.

PubMed, 2018. PubMed Central. [WWW document] `https://www.ncbi.nlm.nih.gov/pmc/` (accessed 19th July 2018).

Rao, Y., Li, Q., Mao, X., Wenyin, L., 2014. Sentiment Topic models for Social Emotion Mining. Information Sciences 266, 90–100.

de Rooij, O., Vishneuski, A., de Rijke, M., 2012. xTAS: Text Analysis in a Timely Manner, in: Proceedings of the 12th Dutch-Belgian Information Retrieval Workshop (DIR 2012), ACM. pp. 89–90.

Schouten, K., Baas, F., Bus, O., Osinga, A., van de Ven, N., van Loenhout, S., Vrolijk, L., Frasincar, F., 2016a. Aspect-Based Sentiment Analysis Using Lexico-Semantic Patterns, in: Proceedings of the 17th International Conference on Web Information Systems Engineering (WISE 2016), Springer. pp. 35–42.

Schouten, K., Frasincar, F., 2016. Survey on Aspect-Level Sentiment Analysis. IEEE Transactions on Knowledge and Data Engineering 28, 813–830.

Schouten, K., Frasincar, F., 2018. Ontology-Driven Sentiment Analysis of Product and Service Aspects, in: Proceedings of the 15th Extended Semantic Web Conference (ESWC 2018), Springer. pp. 608–623.

Schouten, K., Frasincar, F., Dekker, R., 2016b. An Information Gain-Driven Feature Study for Aspect-Based Sentiment Analysis, in: Proceedings of the 21st International Conference on Applications of Natural Language to Information Systems (NLDB 2016), Springer. pp. 48–59.

Schouten, K., Frasincar, F., de Jong, F., 2017a. COMMIT at SemEval-2017 Task 5: Ontology-based Method for Sentiment Analysis of Financial Headlines, in: Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval 2017), Association for Computational Linguistics. pp. 883–887.

Schouten, K., Frasincar, F., de Jong, F., 2017b. Ontology-Enhanced Aspect-Based Sentiment Analysis, in: Proceedings of the 17th International Conference on Web Engineering (ICWE 2017), Springer. pp. 302–320.

Schouten, K., van der Weijde, O., Frasincar, F., Dekker, R., 2018. Supervised and Unsupervised Aspect Category Detection for Sentiment Analysis with Co-occurrence Data. IEEE Transactions on Cybernetics 48, 1263–1275.

Schumaker, R.P., Jarmoszko, A.T., Labedz, C.S., 2016. Predicting wins and spread in the premier league using a sentiment analysis of twitter. Decision Support Systems 88, 76 – 84.

Schumaker, R.P., Zhang, Y., Huang, C.N., Chen, H., 2012. Evaluating Sentiment in Financial News Articles. Decision Support Systems 53, 458 – 464.

Spence, P.R., Lachlan, K.A., Lin, X., del Greco, M., 2015. Variability in twitter content across the stages of a natural disaster: Implications for crisis communication. Communication Quarterly 63, 171–186.

Surdeanu, M., Hicks, T., Valenzuela-Escárcega, M.A., 2015. Two Practical Rhetorical Structure Theory Parsers, in: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL HLT 2015), Association for Computational Linguistics. pp. 1–5.

Takeuchi, H., Subramaniam, L.V., Nasukawa, T., Roy, S., 2009. Getting Insights from the Voices of Customers: Conversation Mining at a Contact Center. Information Sciences 179, 1584–1591.

Taylor, R.N., Medvidović, N., Dashofy, E.M., 2009. Software Architecture: Foundations, Theory, and Practice. John Wiley and Sons.

Usbeck, R., Röder, M., Ngonga Ngomo, A.C., Baron, C., Both, A., Brümmer, M., Ceccarelli, D., Cornolti, M., Cherix, D., Eickmann, B., Ferragina, P., Lemke, C., Moro, A., Navigli, R., Piccinno, F., Rizzo, G., Sack, H., Speck, R., Troncy, R., Waitelonis, J., Wesemann, L., 2015. GERBIL: General Entity Annotation Benchmark Framework, in: Proceedings of the 24th Conference on World Wide Web (WWW 2015), International World Wide Web Conferences Steering Committee. pp. 1133–1143.

Watchmaker, 2018. The watchmaker framework. [WWW document] `http://watchmaker.uncommons.org/` (accessed 19th July 2018).

White, T., 2009. Hadoop: The Definitive Guide. O'Reilly Media, Inc.

Witten, I., Frank, E., 2005. Data Mining: Practical Machine Learning Tools and Techniques. 2nd ed., Morgan Kaufmann.

Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I., 2016. Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM 59, 56–65.