Science of Computer Programming 121 (2016) 128–152

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

TextFlows: A visual programming platform for text mining and natural language processing

Matic Perovšek [a,b,*], Janez Kranjc [a,b], Tomaž Erjavec [a,b], Bojan Cestnik [a,c], Nada Lavrač [a,b,d]

[a] Jožef Stefan Institute, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[c] Temida d.o.o., Ljubljana, Slovenia
[d] University of Nova Gorica, Nova Gorica, Slovenia

ARTICLE INFO

Article history:
Received 14 March 2015
Received in revised form 28 December 2015
Accepted 5 January 2016
Available online 14 January 2016

Keywords:
Text mining
Natural language processing
Web platform
Workflows
Human-computer interaction

ABSTRACT

Text mining and natural language processing are fast growing areas of research, with numerous applications in business, science and creative industries. This paper presents TextFlows, a web-based text mining and natural language processing platform supporting workflow construction, sharing and execution. The platform enables visual construction of text mining workflows through a web browser, and the execution of the constructed workflows on a processing cloud. This makes TextFlows an adaptable infrastructure for the construction and sharing of text processing workflows, which can be reused in various applications. The paper presents the implemented text mining and language processing modules, and describes some precomposed workflows. Their features are demonstrated on three use cases: comparison of document classifiers and of different part-of-speech taggers on a text categorization problem, and outlier detection in document corpora.

1. Introduction

Text mining [9] is a research area that deals with the construction of models and patterns from text resources, aiming at solving tasks such as text categorization and clustering, taxonomy construction, and sentiment analysis. This research area, also known as text data mining or text analytics, is usually considered as a subfield of data mining (DM) research [12], but can be viewed also more generally as a multidisciplinary field drawing its techniques from data mining, machine learning, natural language processing (NLP), information retrieval (IR), information extraction (IE) and knowledge management.

From a procedural point of view, text mining processes typically follow the CRISP-DM reference process model for data mining [7], which proposes six phases when working on a DM project: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. Text mining can be distinguished from general data mining by special procedures applied in the data preparation phase, where unstructured or poorly structured text needs to be converted into organized data, structured as a table of instances (rows) described by attributes (columns). In the modeling phase, such a table of instances can be used by the standard or slightly adapted data mining algorithms to uncover interesting

* Corresponding author.
E-mail addresses: matic.perovsek@ijs.si (M. Perovšek), janez.kranjc@ijs.si (J. Kranjc), tomaz.erjavec@ijs.si (T. Erjavec), bojan.cestnik@temida.si (B. Cestnik), nada.lavrac@ijs.si (N. Lavrač).

http://dx.doi.org/10.1016/j.scico.2016.01.001
0167-6423/

information hidden in the data. Two typical approaches are using clustering algorithms to find groups of similar instances or classification rule learning algorithms to categorize new instances.

The TextFlows platform described in this paper is a new open source, web-based text mining platform that supports the design and composition of scientific procedures implemented as executable workflows. As a fork of ClowdFlows [28], TextFlows has inherited its service-oriented architecture that allows the user to utilize arbitrary web services as workflow components. TextFlows is oriented towards text analytics and offers a number of algorithms for text mining and natural language processing. The platform is implemented as a cloud-based web application and attempts to overcome various deficiencies of similar text analytics platforms, providing novel features that should be beneficial to the text mining community. In contrast to existing text analytics workflow management systems, the developed platform is the only one with all the following properties. It is simple (i.e., enables visual programming, is web-based and requires no installation), enables workflow sharing and reuse, and is open source. Moreover, the platform enables combining workflow components (called "widgets") from different contexts (e.g., using clustering in relational data mining) and from different software sources (e.g., building ensemble classifiers from different libraries). To do so, it provides a unified input-output representation, which enables interoperability between widget libraries through automated data type conversion. It uses a common text representation structure and advocates the usage of 'hubs' for algorithm execution.

TextFlows is publicly available at http://textflows.org, while its source code is available at https://github.com/xflows/textflows under the MIT License. Detailed installation instructions are provided with the source code. After setting up a local TextFlows instance, advanced users can also implement and test their own algorithms. Improvements to the code can also be pushed to the main Git code repository via pull requests. The committed changes are reviewed by the TextFlows core team and merged into the master branch.

TextFlows is a web application which can be accessed and controlled from anywhere while the processing is performed in a cloud of computing nodes. TextFlows differs from most comparable text mining platforms in that it resides on a server (or cluster of machines) while its graphical user interface for workflow construction is served as a web application through a web browser. The distinguishing feature is the ease of sharing and publicizing the constructed workflows, together with an ever growing roster of reusable workflow components and entire workflows. As not only widgets and workflows, but also data and results can be made public by the author, TextFlows can serve as an easy-to-access integration platform both for various text mining workflows but also for experiment replicability. Each public workflow is assigned a unique URL that can be accessed by anyone to either repeat the experiment, or to use the workflow as a template to design another, similar, workflow.

Workflow components (widgets) in TextFlows are organized into packages which allows for easier distributed development. The TextFlows packages implement several text mining algorithms from LATINO[1] [11], NLTK[2] [3] and scikit-learn[3] [32] libraries. Moreover, TextFlows is easily extensible by adding new packages and workflow components. Workflow components of several types allow graphical user interaction during run-time and visualization of results by implementing views in JavaScript, HTML or any other format that can be rendered in a web browser (e.g., Flash, Java Applet).

The rest of the paper is structured as follows. Section 2 presents the technical background and implementation details of the TextFlows platform, along with its key text mining components. The architecture of the system is presented in detail along with specific data structures that allow efficient text mining in a workflow environment. The concept of workflows, their implementation, execution and sharing are presented in Section 3, while Section 4 describes the widget repository and the implemented modules of the platform. The advanced features of TextFlows are demonstrated in Section 5 on three use cases: a comparison of document classifiers on a classification problem, a comparison of part-of-speech taggers on a text categorization (classification) problem, and outlier detection in document corpora. Section 6 presents the related work, where we describe comparable text mining platforms together with their similarities and differences compared to the TextFlows platform. Section 7 concludes the paper by presenting a summary and some directions for further work.

2. The TextFlows platform

This section presents the TextFlows platform, together with its architecture and main components of the system. We also introduce the graphical user interface and describe the concept of workflows. Like its predecessor data mining platform ClowdFlows [28], TextFlows can also be accessed and controlled from a browser, while the processing is performed on a cloud of computing nodes. In this section we explain the relationship between TextFlows and ClowdFlows, present the architecture of the TextFlows platform and describe the key text mining concepts of TextFlows in more detail.

2.1. Platform architecture

In software engineering, terms *front-end* and *back-end* are used to distinguish the separation between a presentation layer (the client side) and a data access layer (the server side), respectively. Fig. 1 shows the TextFlows architecture, which

[1] LATINO (Link Analysis and Text Mining Toolbox) is open-source—mostly under the LGPL license—and is available at http://source.ijs.si/mgrcar/latino.
[2] http://www.nltk.org.
[3] http://scikit-learn.org.

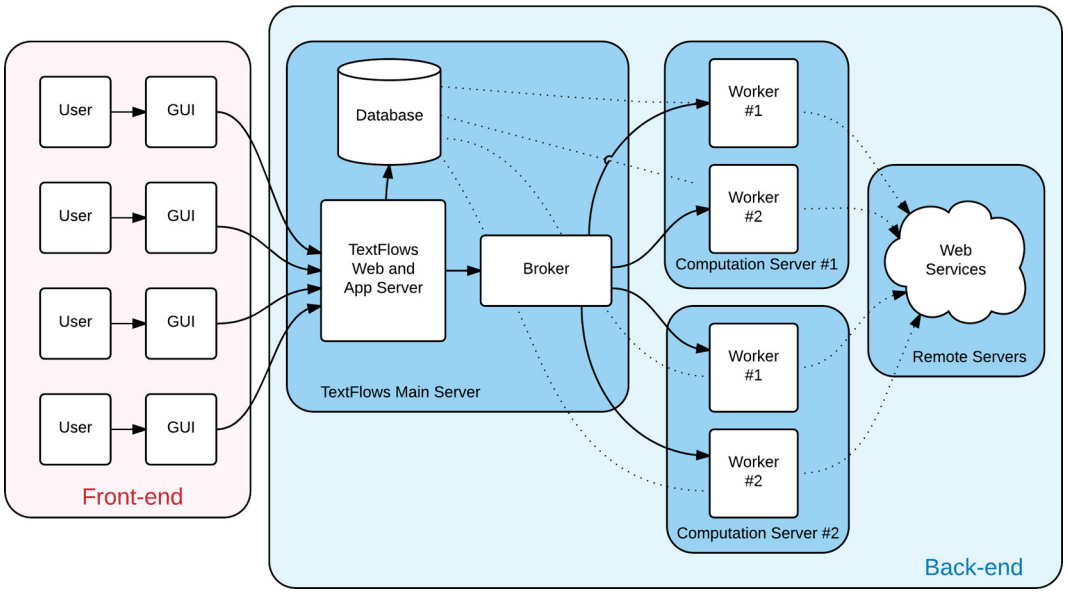

Fig. 1. An overview of the TextFlows architecture, separated into the front-end (in pink) and the back-end (in blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

is logically separated into the front-end (in pink) and the back-end (in blue). The back-end comprises a relational database for storing workflows, workers for task execution and a broker for delegating tasks from different queues to workers which can reside on different clusters of machines. The front-end is designed for user interaction with workflows through the graphical user interface in a web browser.

The back-end of the TextFlows platform uses Django,[4] a Python-based open source high-level web framework. Django follows the model–view–controller architectural design, which encourages rapid development, and includes an object-relational mapper and a template engine for efficient code generation. The object-relational mapper provides an interface which links objects to a database. This provides support for several database systems, such as PostgreSQL, MySQL, SQLite and Oracle databases. PostgreSQL is used in the public installation of TextFlows.

Without additional extensions Django is synchronous, sometimes also described as blocking. This means that a HTTP request will not be returned until all processing is complete. Though this is the expected behavior usually required in web applications, in the case of TextFlows we need tasks to run in the background without blocking. Furthermore, different system environment requirements of implemented libraries dictate that the TextFlows platform must be distributed across multiple machines (e.g., the LATINO library uses the. Net framework and performs best on Windows operating systems). TextFlows task queues are used as a mechanism to distribute work across threads and machines. This is performed via Celery,[5] which is a task queue system based on distributed message passing. Celery is focused on real-time operation, but supports scheduling as well. Dedicated worker processes monitor task queues for new work to perform and active workers execute different tasks concurrently on multiple servers. Tasks can be executed asynchronously (in the background) or synchronously (wait until ready). A Celery system can consist of multiple workers and brokers, thus supporting high availability and horizontal scaling.

Celery communicates via messages and uses a message broker to mediate between clients and workers. To initiate a task, a client adds a message to the queue, which the broker then delivers to a worker. The system used as a broker in TextFlows is RabbitMQ,[6] a complete and highly reliable enterprise messaging system based on the Advanced Message Queuing Protocol (AMQP). It offers not only exceptionally high reliability, but also high availability and scalability, which is vital for the TextFlows platform.

TextFlows uses the PySimpleSoap library[7] for integrations of web services as workflow components. PySimpleSoap is a lightweight Python library which provides an interface for client and server web service communication. Using PySimpleSoap we can not only import WSDL web services as workflow components, but also expose entire workflows as WSDL web services.

The client side of the TextFlows platform consists of operations that involve user interaction primarily through the graphical user interface (GUI) in a modern web browser. The graphical user interface is implemented in HTML and JavaScript,

[4] https://www.djangoproject.com.
[5] http://www.celeryproject.org.
[6] http://www.rabbitmq.com.
[7] https://code.google.com/p/pysimplesoap/.

with an extensive use of the jQuery library.[8] The jQuery library was designed to simplify client-side scripting and is the most popular JavaScript library in use today.[9] On top of jQuery we use the interaction library jQuery-UI,[10] which is a collection of GUI modules, animated visual effects, and themes. This library supports the option to make elements in the graphical user interface draggable, droppable, and selectable, which are the features supported by the TextFlows workflow canvas (cf. Section 3).

2.2. Key text mining concepts in TextFlows

The key concepts used in text mining and natural language processing are a document collection (or corpus), a single document (or text), and document features (or annotations) [9]. The following sections describe the model of corpora, documents and annotations in TextFlows. When designing TextFlows, emphasis was given to common representations that are passed among the majority of widgets: each TextFlows document collection is represented by an instance of the *AnnotatedDocumentCorpus (ADC)* class, a single text is an instance of the *AnnotatedDocument* class, while the features are instances of the *Annotation* class.

2.2.1. Annotated corpus

A document collection is any grouping of text documents that can be used in text analytics. Even though the size of a collection may vary from a few to millions of documents, from the text analysis perspective, more is better. In TextFlows, the Python class that represents a corpus of documents is called *AnnotatedDocumentCorpus (ADC)*. Every ADC instance contains not only a collection of documents which are part of this corpus but also the features that provide additional information about the corpus (e.g., authors, date of collection, facts and notes about the dataset, etc.). The features are stored in a simple key-value Python dictionary, where keys are strings and the values can store any Python object.

2.2.2. Annotated document

In TextFlows a single textual data unit within a collection—a document—is represented by the class *AnnotatedDocument*. An *AnnotatedDocument* object contains the text of the entire document, which may vary in size, e.g., from a single sentence to a whole book.

Similarly to *AnnotatedDocumentCorpus*, *AnnotatedDocument* instances in TextFlows also contain features which may provide information about a single document (e.g., author, date of publication, document length, assigned keywords, etc.).

2.2.3. Annotation

In TextFlows *Annotation* instances are used to mark parts of the document, e.g., words, sentences or terms. Every *Annotation* instance has two pointers: one to the start and another to the end of the annotation span in the document text. These pointers are represented as the character offset from the beginning of the document. *Annotation* instances also have a type attribute, which is assigned by the user and is used for grouping annotations of similar nature. As described in detail in Section 4, annotations can also contain key-value dictionaries of features, which are used by various taggers to annotate parts of document with a specific tag, e.g., annotations of type “token” that have a feature named “StopWord” with value “true”, represent stop words in the document.

3. The concept of workflows

The workflow model in the TextFlows platform consists of an abstract representation of workflows and workflow components. A workflow is an executable graphical representation of a complex procedure. The graphical user interface used for constructing workflows follows a visual programming paradigm which simplifies the representation of complex procedures into a spatial arrangement of building blocks. The most basic unit component in a TextFlows workflow is a processing component, which is represented as a widget in the graphical representation. Considering its inputs and parameters every such component performs a task and outputs the results. Different processing components are linked via connections through which data is transferred from a widget’s output to another widget’s input. Additional inputs for a widget are its parameters, which the user enters into the widget text fields. The graphical user interface implements an easy-to-use way of arranging widgets on a canvas to form a graphical representation of a complex procedure.

The TextFlows graphical user interface, illustrated in Fig. 2, consists of a widget repository and a workflow canvas. The widget repository is a set of widgets ordered in a hierarchy of categories. Upon clicking on a widget in the repository, the widget is added as a building block to the canvas. While hovering over a widget its documentation is shown in as a tooltip. Connections between widgets can be added by clicking on an output of a widget and then on an input of another widget. The workflow canvas implements moving, connecting, and issuing commands to execute or delete widgets. Every action on the workflows canvas causes an asynchronous HTTP POST request to be sent to the server. After the requested operation is

[8] http://jquery.com.
[9] http://w3techs.com/technologies/overview/javascript_library/all.
[10] http://jqueryui.com.

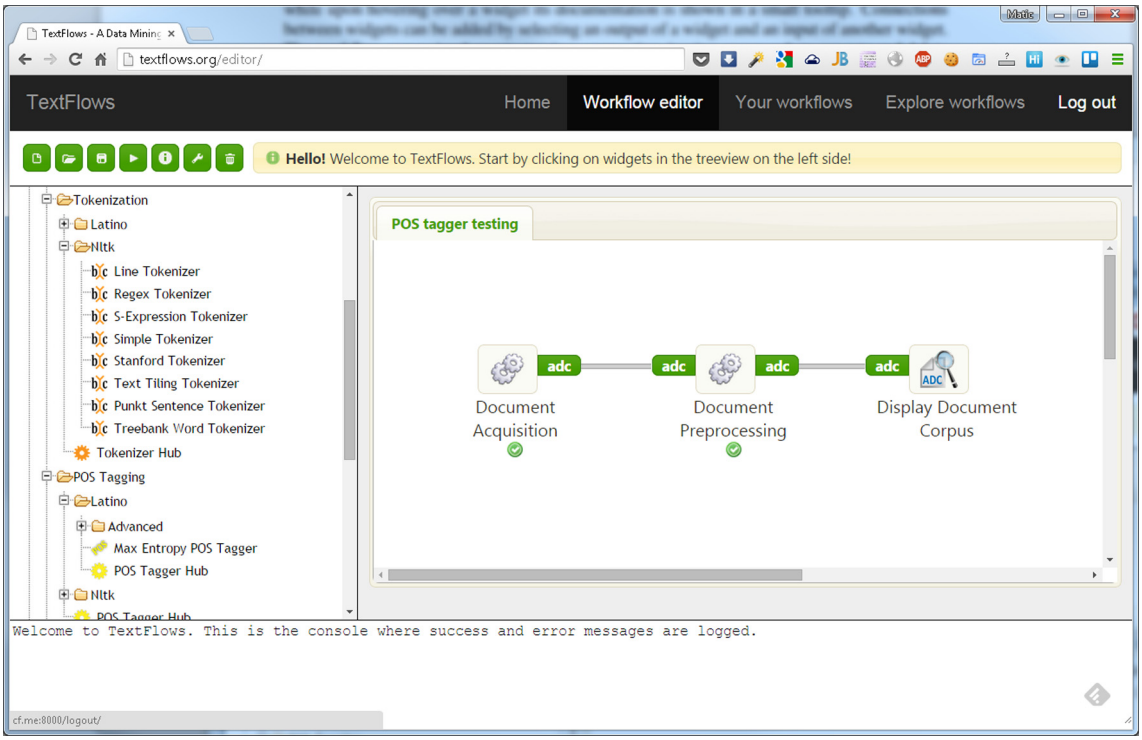

Fig. 2. A screenshot of the TextFlows GUI opened in Google Chrome. On the top there is a toolbar for saving, deleting entire workflows. On the left is the widget repository giving a list of available widgets grouped by their functionality. By clicking on a widget in the repository it is added to the workflow construction canvas which is on the right. The console for displaying success and error message is located on the bottom of the interface.

validated on the server, a success or error message with additional information is passed to the user interface. An example of such a validation is checking for cycles in the workflows when connecting widgets.

3.1. Workflow execution

The workflow execution engine is responsible for executing the workflow widgets in the predefined order. Two such engines are integrated into the TextFlows platform, one implemented in Python and another in JavaScript. Sub-parts of workflows in subprocesses and loops are executed by the Python implementation, while the top-level workflows executed from the user interface (when the user actually needs to see the order of the executed widgets in real time) are processed by the JavaScript implementation. The former shows results only after their complete execution, while the later allows showing results of the execution in real time. With the Python implementation the server receives only one HTTP request for the entire part of the workflow, therefore multiprocessing had to be implemented manually. On the other hand, when a workflow is running with the JavaScript engine, it perpetually checks for widgets that are executable and executes them. Executable widgets are widgets which either have no predecessors or their predecessors have already been successfully executed. Whenever two or more independent widgets can be executed at the same time they are asynchronously executed in parallel. Each widget is executed through a separate asynchronous HTTP request. Every request is handled by the server separately and executes a single widget. When a widget is prepared to be executed, a task is added to a suitable task queue; as some widgets have special library requirements or even system requirement they are executed on a separated task queue with its dedicated workers. Celery communicates via messages and uses a message broker to find a suitable worker to which the task is delivered to. Dedicated worker processes monitor task queues for new work to perform. When the task is executed its result is saved into the database and returned to the client. The execution of a workflow is considered complete when there are no executable or running widgets.

3.2. Workflow sharing

Workflows in TextFlows are processed and stored on remote servers from where they can be accessed from anywhere, requiring only an Internet connection. By default each workflow can only be accessed by its author, although (s)he may also choose to make it publicly available. The TextFlows platform generates a specific URL for each workflow that has been saved as public. Users can then simply share their workflows by publishing the URL. Whenever a public workflow is accessed by another user, a copy of the workflow is created on the fly and added to their private workflow repository. The workflow

is copied with all the data to ensure that the experiments can be repeated. In this way, the users are able to tailor the workflow to their needs without modifying the original workflow.

4. The widget repository

This section presents the TextFlows widget repository. First we describe different types of widgets, followed by the presentation of widgets based on their functionality as they appear in the TextFlows widget repository.

Widgets in TextFlows are separated into four groups based on their type:

- *Regular widgets* perform specific tasks that transform the data from their inputs and parameters to data on their outputs, and provide success or error messages to the system. In the back-end such widgets are represented as Python functions, which receive (on every widget execution) a Python dictionary of inputs and parameters as widget arguments, perform a specific task and return a dictionary of outputs. The function is called each time the widget is executed. Widgets that implement complex long-running procedures can also display a progress bar, which shows the progress to the user in real time.
- *Visualization widgets* are extended versions of regular widgets as they also provide the ability to render an HTML template with JavaScript to the client's browser, which is useful for data visualizations and presentation of a more detailed feedback to the user. Visualization widgets differ from regular widgets by a secondary Python function which controls the rendering of the template. This function is only invoked when the widget is executed using the JavaScript execution engine, i.e., when it is not part of a subprocess.
- *Interactive widgets* are extensions of regular widgets as they pop-up an additional window during execution through which the user can interact with or manipulate the data (an example of an interactive widget is shown in Fig. 12). The entire procedure is implemented using three Python functions. The first function receives the widget's inputs and initialization parameters as its arguments and prepares the data for the second function. The second function renders (using an HTML template) a pop-up window that prompts the user to manipulate the data. The final function uses the user inputs, as well as the widget's inputs and parameters to produce the final output of the widget.
- *Workflow control widgets* provide additional workflow controls which allow the user to combine different workflow components into subprocesses, and provide different types of iterations through data (e.g., iteration through a list of classifiers, applying a classifier to all folds in cross-validation, etc.). This group of widgets consists of: *Subprocess*, *Input*, *Output*, *For Input*, *For Output*, *Cross Validation Input* and *Cross Validation Output* widgets. Whenever a *Subprocess* widget is added to a workflow, an initially empty workflow with no inputs and outputs is created. Then, when an *Input* or *Output* widget is attached to a subprocess workflow, an input or output is automatically added to the subprocess widget itself. Workflows can be indefinitely nested this way.

  Two additional variations of the input and output widgets exist in TextFlows. When a subprocess contains the *For Input* and *For Output* widgets, the workflow execution engine will emulate a for loop by attempting to break down the object on the input and executing the subprocess workflow once on every iteration. Using these controls a subprocess can be iteratively executed on a list. Similarly, if the user opts to use the *Cross Validation Input* and *Cross Validation Output* widgets the input data will be divided into the training and test dataset according to the selected number of folds; if the input data is labeled, stratified cross-validation [40] is performed.

The widget repository shows the hierarchy of all the widgets currently available in the TextFlows platform, grouped by their functionality. There are four top-level categories:

- *Text mining widgets:* a collection of implemented text mining widgets; these widgets are further divided based on their text mining functionality.
- *Basic widgets:* widgets that are responsible for creating and manipulating simple objects such as strings and integers.
- *Subprocess widgets:* a collection of workflow control widgets, which are required for visual programming of complex procedures.
- *WSDL imports:* workflow components representing the WSDL web-services that the user has imported.

In the following sections we present in more detail the text mining widgets based on their functionality in the order of appearance in the TextFlows widget repository.

4.1. Corpus and vocabulary acquisition

Document acquisition (import) is usually the first step of every task, where TextFlows employs various widgets to enable loading document corpora, labeling of documents with domain labels and converting them into the *AnnotatedDocumentCorpus* structure. We identified the following text document acquisition scenarios, which are also supported by the developed widgets:

- *Loading locally stored files in various application dependent formats*. In TextFlows document corpora can be uploaded from local files using the *Load Document Corpus* interactive widget. The entire dataset can be either a single text file (.doc, .docx, .pdf file formats are supported), where each line represents a separate document, or a zip of files in which a document is represented as a file. Apart from text, the files may optionally contain document titles, as well as multiple labels, which are encoded by the first word within a document prefixed by an exclamation mark, e.g., "!positive" is used to denote that the document belongs to the "positive" document category.
- *Acquiring documents using the WSDL+SOAP web services*. The user can integrate third-party web services as workflow components using the *Import webservice* button obtained from the bottom of the widget repository. Such integration allows for the inclusion of database web services into the workflow (e.g., PubMed offers a SOAP web service interface to access their database). TextFlows currently supports WSDL as the interface definition language for describing connections and requests to the web service and SOAP as the format for sending and receiving messages. The output of the imported web service widget can be connected to the *Load Document Corpus* widget that transforms the plain text input documents into the *AnnotatedDocumentCorpus* structure.
- *Selecting documents from SQL databases*. The TextFlows platform supports loading data only from MySQL databases via the *Load Document Corpus from MySQL* widget. Before execution the user enters the information which is required to connect to a database (e.g., user credentials, database address, database name, table name, column names, etc.) in order to retrieve the data from a MySQL database server. This widget then connects to the specified MySQL database server and returns the input columns representing document titles, texts and labels from the selected table. The final output of the widget is an automatically constructed *Annotated Document Corpus* object.
- *Crawling the Internet for gathering documents from web pages*. The *Crawl URL links* widget receives as an input a list of URLs, e.g., Wikipedia pages. First, every page is visited by an automatic crawler in order to gather the website's (HTML) content. Next, the Boilerpipe library [27] is used to extract the linguistically relevant textual content from the web page source. There are several content extraction methods available which can be selected by the user from the widget's properties. Finally, the *Crawl URL links* outputs the *Annotated Document Corpus* where document titles are represented with URLs and the extracted website texts become the document texts.
- *Collecting documents from snippets returned from web search engines*. In TextFlows the user can search the web using the *Search with Bing* and *Search with Faroo* widgets, which use the Microsoft Bing[11] and Faroo[12] as their web search engines, respectively. Both widgets require the user to enter the search query as well as the number of search results that the widget should return. The output of both widgets is a list of URLs which are returned by the web search engine. The execution the output can be connected to the *Crawl URL links* widget which will extract the web content for every URL.

The most straightforward technique to incorporate background knowledge about the documents and their domain is to use a controlled vocabulary. A controlled vocabulary is a lexicon of all terms that are relevant for a given domain. TextFlows allows the users not only to upload their own local vocabulary files but also gives them the possibility to use one of the implemented vocabulary construction tools, such as the *MeSH filter* widget which constructs a vocabulary containing all the terms that belong to the user selected descriptors from the MeSH hierarchy.[13]

4.2. Corpus manipulation and visualization

TextFlows implements widgets which allow the manipulation of *AnnotatedDocumentCorpus* (ADC) data objects. They allow the user to add new features, extract existing features from a document corpus, split document corpora (by either specifying conditions or by indices), merge different corpora, etc.

A special widget in the platform is *Document Corpus Viewer*, which visualizes the ADC data objects (note that TextFlows emphasizes the importance of the common ADC representation which is passed among the majority of widgets). As shown in Fig. 3, the *Document Corpus Viewer* interactive widget allows the user to check the results of individual widgets by visualizing the ADC data object from their outputs. Through its interactive interface the widget allows the user to select an individual document from the list of document snippets by simply clicking on it. This opens a new page with a detailed view of the selected document, as shown in Fig. 3.

4.3. Text preprocessing widgets

Preprocessing is a very important part of any form of knowledge extraction from text documents. Its main task is the transformation of unstructured data from text documents into a predefined well-structured data representation. In general, the task of preprocessing is to construct a set of relevant features from text documents. The set of all features selected for a given document collection is called a representational model [42,43,9]. Each document is represented by a vector of

[11] http://www.bing.com/.

[12] http://www.faroo.com/.

[13] MeSH (Medical Subject Headings) is a controlled vocabulary thesaurus used for indexing articles in PubMed, a database designed by The National Library of Medicine. The MeSH database is available online http://www.nlm.nih.gov/mesh.

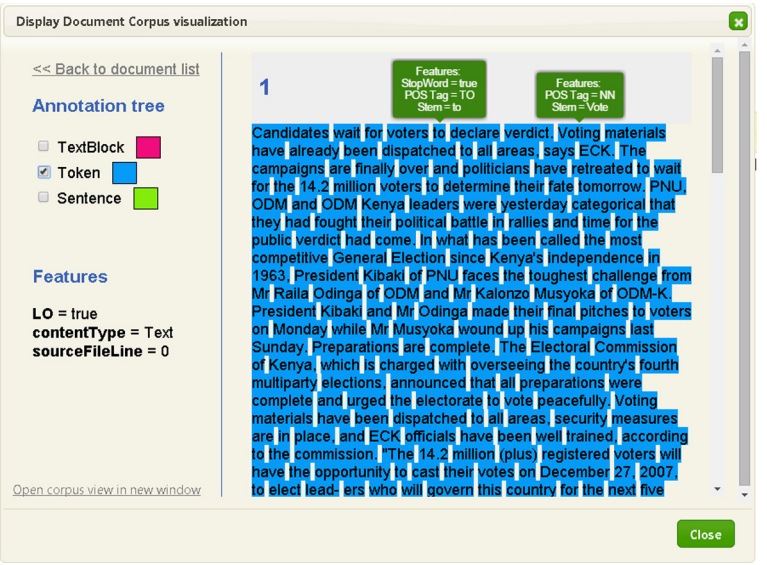

Fig. 3. Document Corpus Viewer widget visualizes the *AnnotatedDocumentCorpus* data objects. This figure shows the document exploration page which displays a detailed view for a selected document. On the left side a list of applied tokenizations is shown, while on the right the document's full text is displayed. When the user selects a tokenization from the list, the tokens are highlighted directly on the text of the document with an alternative background color. As shown in the figure, annotation features are shown by hovering over a token.

numerical values, one for each feature of the selected representational model. Using this construction, we get the most standard text mining document representation, called feature vectors, where each numeric component of a vector is related to a feature and represents a weight related to the importance of the feature in the selected document. Typically, such text-based feature vectors are very sparse, as the majority of weights are equal to zero [9]. The goal of text preprocessing widgets is to extract a high quality feature vector for every document in a given document corpus.

Technically, our implementation employs the LATINO (Link Analysis and Text Mining Toolbox) software library [11], NLTK (Natural Language Toolkit) library [3] and scikit-learn library [32] for the text preprocessing needs. Together they contain the majority of elementary text preprocessing procedures as well as a large number of advanced procedures that support the conversion of a document corpus into a table of instances, thus converting every document into a table row representation of an instance.

In the TextFlows widget repository preprocessing techniques are based on standard text mining concepts [9] and are implemented as separate categories. Every category possesses a unique hub widget, which has the task of applying an given preprocessing technique from its category to the *AnnotatedDocumentCorpus* data object. Every such widget is library independent, meaning that it can execute objects from either LATINO, NTLK or scikit-learn libraries.

A standard collection of preprocessing techniques is listed below, together with sets of functionalities implemented in our platform:

1. *Tokenization*. In tokenization, meaningful tokens are identified in the character stream of the document, such as words or terms. TextFlows offers a large set of tokenizers: from LATINO, such as *Max Entropy Tokenizer* (word and sentence), *Unicode, Simple* and *Regex* tokenizers; various tokenizers from the NLTK library, from simpler ones, such as *Line, Regex, S-Expression, Simple*, to more complex ones, like *Stanford Tokenizer* and *Treebank Word Tokenizer*. Every tokenizer can be applied on a document corpus using the *Tokenizer Hub* widget. This hub receives as an input an ADC data object and a tokenizer instance, as well as two parameters entered by the user: the type of annotations to be tokenized (e.g., "TextBlock") and the type of annotations to be produced (e.g., "Sentence", "Token"). The *Tokenizer Hub* finds annotations of the input type and tokenizes them using the input tokenizer. The output of the hub is a new ADC object, which now contains the annotations of the new type. As described in the previous section, the results of any corpus tokenization can be visualized using the *Display Document Corpus* widget, as shown in Fig. 3.
2. *Stop word removal*. Stop words are predefined words from a language that do not carry relevant semantic information (e.g., articles, prepositions, conjunctions, etc.); the usual practice is to ignore them when building a feature set. In TextFlows we have three widgets which are used for stop word tagging: *Get StopWord Set* (outputs a predefined list of stop words for the user selected language—stop word lists for 18 languages, taken from Snowball,[14] are included in our library), *StopWords Tagger* (receives as an input a list of stop words and outputs a constructed tagger object, which tags the words from the input list as stopwords), *StopWord Tagger Hub* (responsible for applying a stop word tagger on a

[14] Snowball: A small string processing language designed for creating stemming algorithms: http://snowball.tartarus.org/.

document corpus). Similarly to the *Tokenization Hub*, the *Stop Word Tagger Hub* receives on its inputs an ADC data object and a stop word tagger instance. The user is able to enter two parameters: the type of annotations to be tagged (as a stop word) and a feature name, which is added (with a default value of 'true') to every annotation of the selected type, which the tagger marks as a stop word. The output of the hub is a new ADC object. Fig. 3 shows the visualization of a selected document from the output ADC data object using the *Display Document Corpus* widget. The stop word annotation features are shown by hovering over the documents tokens.

3. *Part-of-speech tagging*. Tagging annotates words with the appropriate part-of-speech (PoS) tags based on the context in which they appear. The TextFlows platform includes the LATINO's *Max Entropy PoS Tagger* and from NLTK the following taggers: *Affix PoS Tagger* (learns only on term affixes), *Brill's rule-based PoS Tagger* [5], *Classifier-based PoS Tagger* (requires a classifier and a pre-annotated dataset to learn the PoS tags) and a *PoS N-gram tagger* (learns on a pre-annotated dataset the most frequent tag for every n-gram). PoS tags are applied to ADC data using the *PoS Tagger Hub*. The *PoS Tagger Hub* requires, besides the usual element annotation type (default: "Token") and the PoS feature name (default: "PoS Tag"), an additional parameter input by the user: a sentence annotation type (default: "Sentence"). The hub tags element annotations in the context of sentence annotations by assigning them new features with values returned by the PoS tagger. Fig. 3 visualizes a selected document from the output ADC data object using the *Display Document Corpus* widget. The generated PoS tag features are shown when hovering over the document's tokens.
4. *Stemming and lemmatization*. This is the process of converting words/tokens into their stem or citation forms. The following stemmers/lemmatizers were taken from the LATINO library: *Stem Tagger Snowball* and the *Lemma Tagger LemmaGen* [23]. We have also implemented the following widgets which represent the corresponding algorithms from the NLTK library: *Porter Stemmer*, *Snowball Stemmmer*, *ISRI Stemmer*, *Regex Stemmer*, *RSLP Stemmer*, *Lancaster Stemmer*, and *WordNet Lemmatizer*. Analogous as in the stop word removal category, stemmers and lemmatizers can be applied using the *Stem/Lemma Tagger hub*. This widget receives as an input an ADC data object and a stemmer (or lematizer) instance and outputs a new ADC data object with an additional stemming added. The user can enter two parameters: the type of annotations to be stemmed ("Token" by default) and a feature name ("Stem" by default), which will be assigned to every annotation of the selected type as a key-value pair together with the stemmed value.

4.4. Bag-of-Words model

In the most general case, when dealing with raw text, the features are derived from text using only text preprocessing methods. The most common document representation model in text mining is the Bag-of-Words (BoW) model [9]. It uses all words (or, e.g., terms) as features, therefore the dimension of the feature space is equal to the number of different words in all of the documents. One of the most important characteristics of the described document features is the fact that they are usually very sparse [9], meaning that most of the features in a vector have zero weight. This sparsity is due to the fact that there are many different features (words, terms, concepts) in the document corpus; yet, a single document contains only a small subset of them. Consequently, the resulting feature matrix will have many (typically more than 99%) feature values that are zeros.

The TextFlows platform uses the Compressed Sparse Row (CSR) matrices, implemented in the scipy.sparse package[15] in order to be able to efficiently store the matrix of features in memory and also to speed up algebraic operations on vectors and matrices. The CSR matrices make no assumptions about the sparsity structure of the matrix, and do not store any unnecessary elements. Their main purpose is to put the subsequent non-zeros of the matrix rows in contiguous memory locations. Usually three vectors are created: one for storing floating-point numbers (*values*), and the other two for integers (*col_ ind, row_brk*). The values vector stores the values of the non-zero elements of the matrix, as they occur if reading through the matrix row by row. The *col_ind* vector stores the column indexes of the elements in the val vector, while the *row_brk* vector stores the locations in the *values* vector that start a new row in the dense original matrix. The storage savings using this approach are significant. Instead of storing $m * n$ elements, we only need to use $2 * \text{nnz} + m$ storage locations, where m is the number of rows, n is the number of columns and nnz is the number of non-zeros in the dense matrix.

In the data mining modeling phase (i.e., document classification or text clustering), each document from the ADC structure needs to be represented as a set of document features it contains. In TextFlows the *Construct BoW Dataset and BoW Model Constructor* widget takes as an input an ADC data object and generates a sparse BoW model dataset (which can be then handed to a classifier). The widget takes as an input also several user defined parameters, which are taken into account when building the feature dataset:

- *Token Annotation.* This is the type of *Annotation* instances marking parts of the document (e.g., words, sentences or terms), which will be used for generating the vocabulary and the dataset.
- *Feature Name.* If present, the model will be constructed out of annotations' feature values instead of document text. For example, this is useful when we wish to build the BoW model using stems instead of the original word forms.

[15] http://docs.scipy.org/doc/scipy/reference/sparse.html.

- *Stop Word Feature Name.* This is an annotation feature name which is used to tag tokens as stop words. These tokens will be excluded from the BoW representational model. If the stop word feature name in not provided all tokens are included in the BoW space.
- *Label Document Feature Name.* This is the name of the document feature which will be used as a class label of the examples in the dataset. If blank, the generated sparse dataset will be unlabeled.
- *Maximum n-gram Length.* The upper bound of the range of n-values for different n-grams to be extracted. All values of n such that $1 \leq n \leq$ max_ngram will be used.
- *Minimum Word Frequency.* Cut-off frequency value for including an item into the vocabulary.
- *Word Weighting Type.* The user can select among various weighting models for assigning weights to features:
  - Binary. A feature weight is 1 if the corresponding term is present in the document, or zero otherwise.
  - Term occurrence. A feature weight is equal to the number of occurrences of the corresponding term. This weight is sometimes better than a simple binary value since frequently occurring terms are likely to be more relevant to the given text.
  - Term frequency. A weight is derived from the term occurrence by dividing the vector by the sum of all vector's weights.
  - TF-IDF. Term Frequency-Inverse Document Frequency [42] is the most common scheme for weighting features. For a given term w in document d from corpus D, the TF-IDF measure is defined as follows:

$$\text{tfIdf}(w, d) = \text{tf}(w, d) \times \log \frac{|D|}{|\{d \in D : w \in d\}|}, \tag{1}$$

    where $\text{tf}(w, d)$ represents the number of times term w appears in document d. The reasoning behind the TF-IDF measure is to lower the weight of terms that appear in many documents as this is usually an indication of them being less important (e.g., stop-words). The appropriateness of this scheme was confirmed in numerous text mining problem solutions [14,9].
  - Safe TF-IDF. For a given term w in document d from corpus D, the Safe TF-IDF measure is defined as follows:

$$\text{safeTfIdf}(w, d) = \text{tf}(w, d) \times \log \frac{|D|}{|\{d \in D : w \in d\}| + 1}, \tag{2}$$

    This approach smoothens IDF weights by adding one to document frequencies, as if an extra document was seen containing every term in the collection exactly once. This prevents the occurrence of divisions by zero.
  - TF-IDF with sublinear TF scaling. It often seems unlikely that twenty occurrences of a term in a document truly carry twenty times the significance of a single occurrence. Accordingly, there has been considerable research into variants of term frequency that go beyond counting the number of occurrences of a term [31]. A common modification is to use the logarithm of the term frequency instead of *tf*, defined as:

$$\text{wf}(w, d) = \begin{cases} 1 + \log \text{tf}(w, d), & \text{if } \text{tf}(w, d) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

- *Normalize Vectors.* The weighting methods can be further modified by vector normalization. If the user opts to use it in TextFlows the *L2* regularization [30] is performed.

Besides the sparse BoW model dataset the *Construct BoW Dataset and BoW Model Constructor* also outputs a *BowModelConstructor* instance. This additional object contains settings which allow repetition of the feature construction steps on another document corpus. These settings include the input parameters, as well as the learned term weights and vocabulary.

An important widget in the Bag-of-Words category is the *Create BoW Dataset using BoW Model Constructor*. Its task is to apply the input *BowModelConstructor* instance to an input ADC data object and create a sparse feature dataset. This is useful, for instance, in every cross-validation fold where you need to build the test dataset's sparse matrix using the same settings (also including IDF term weights) used for building the training sparse matrix.

4.5. Document classification

Document classification (also called text categorization) refers to automated assigning of predefined categories to natural language texts. A primary application of text categorization systems is to assign subject categories to documents to support information retrieval, or to aid human indexers in assigning such categories. Text categorization components are also increasingly being used in natural language processing systems for data extraction. Classification techniques have been applied to spam filtering, language identification, genre classification, sentiment analysis, etc. The common approach to building a text classifier is to manually label a selected set of documents to predefined categories or classes, and use them to train a classifier. The trained classifier can then be used to assign class labels to unseen documents based on the words they contain.

The term *supervised learning* refers to the above-described approach to automatically building a text classifier from training documents, which have been labeled (often manually) with predefined classes. The TextFlows platform currently

contains only the supervised learning approaches from the LATINO, NTLK and scikit-learn libraries. Every widget contains input parameters which are used to construct the classifier object. From the LATINO library we integrated *Nearest Centroid Classifier, Naive Bayes Classifier, SVM (Binary and Multiclass) Classifier, Majority Classifier, Maximum Entropy Classifier, kNN (fast and regular version) Classifier*, while the NLTK library contributes an additional Naive Bayes Classifier. The following widgets represent classifiers from the scikit-learn library: *Decision Tree Classifier, Multinomial Naive Bayes Classifier, Gaussian Naive Bayes Classifier, k-Nearest Neighbors Classifier, SVM Linear Classifier, SVM Classifier*

For training and applying classifiers TextFlows offers two dedicated widgets: *Train Classifier Hub* and *Apply Classifier Hub*. The *Train Classifier Hub* receives on its inputs a sparse feature dataset object and an untrained classifier. Its function is to fit the classifier model according to the given training data. The final outcome of this widget is a trained classifier object.

The *Apply Classifier Hub* receives a trained classifier object and returns predicted class probabilities for every new document from the input test dataset, as well as the test dataset with a new predicted labels column.

4.6. Literature-based discovery

Literature-based discovery refers to the use of papers and other academic publications (the "literature") to find new relationships between existing knowledge (the "discovery"). [48] presented an approach to discovering unknown relations between previously unrelated concepts by identifying interesting bridging terms (b-terms) appearing in two separate document sets and bearing the potential to indirectly connect the two concepts under investigation.

The TextFlows platform includes a dedicated category with several widgets which support literature-based discovery. For example, literature-based discovery is supported through the incorporated *Explore In CrossBee* widget, which provides a link to the web application CrossBee (Cross-Context Bisociation Explorer) developed by [21]. The web application can be used for cross-domain knowledge discovery from a given pair of user-selected document corpora.

4.7. Noise detection

Noise filtering is frequently used in data preprocessing to improve the accuracy of induced classifiers. TextFlows incorporates an ensemble-based noise ranking methodology for explicit noise and outlier identification, named NoiseRank [45], which was modified to work with texts and TextFlows ADC data objects. Its main aim is to detect noisy instances for improved data understanding, data cleaning and outlier identification. NoiseRank was previously successfully applied to a real-life medical problem [45]. We show an example of using the NoiseRank methodology on a task of outlier detection in document corpora in Section 5.3.

4.8. Evaluation and visualization

The TextFlows platform enables users to create interactive charts for easy and intuitive evaluation of performance results. It includes standard performance visualizations used in machine learning, data mining, information retrieval, etc. Notably, the TextFlows platform includes a full methodology, named VIPER [45,46], i.e., a visualization approach that displays the results as points in the two dimensional precision-recall space. The platform contains several visual performance evaluation widgets, which result in interactive performance charts that can be saved and exported to several formats.

- *Scatter charts*. These include ROC space charts and PR space charts.
- *Curve charts*. These include PR curves, Cost curves, Lift curves, ROC curves, Kendall curves and Rate-driven curves.
- *Column charts*. These are general column charts for visualizing multiple performance measures for a set of algorithms.

While VIPER visualizations appear to be straightforward, this visualization toolbox is innovative and very useful for text analytics. An example is visual comparison of F-value results of different text analysis tools, including F-isoline-based text classifier comparison, which is not supported by any other visualization tool. We demonstrate several implemented visualization techniques in Section 5.

5. Selected use cases

In this section we demonstrate advanced features of TextFlows on three use cases: a comparison of classifiers from different libraries for a text categorization problem, a comparison of PoS taggers on the same categorization problem and outlier detection in document corpora.

In our experiments we used a corpus of documents, presented by [35] and [36], which was originally collected by the IPrA Research Center, University of Antwerp. The document corpus contains 464 articles (about 320,000 words) concerning Kenyan presidential and parliamentary elections, held on 27th December 2007, and the crisis following the elections. The documents originate from six different daily newspapers in English, covering the time period from 22nd December 2007 to 29th February 2008. Articles from the US and British press (The New York Times, The Washington, The Independent, The Times and Post) form the class label "Western" (WE) and articles from local Kenyan newspapers Daily Nation and The Standard are categorized as "Local" (LO).

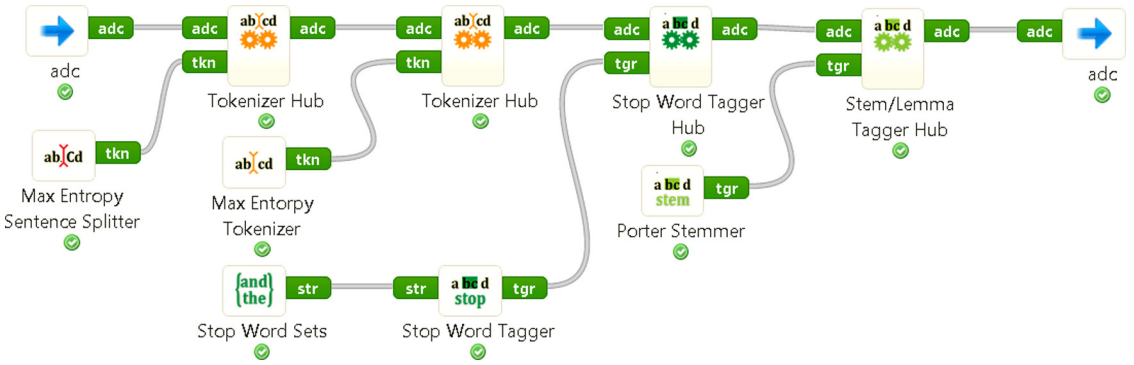

Fig. 4. Document preprocessing workflow, available at http://textflows.org/workflow/604/. The same workflow is implemented as a subprocess in the three use cases presented in this paper.

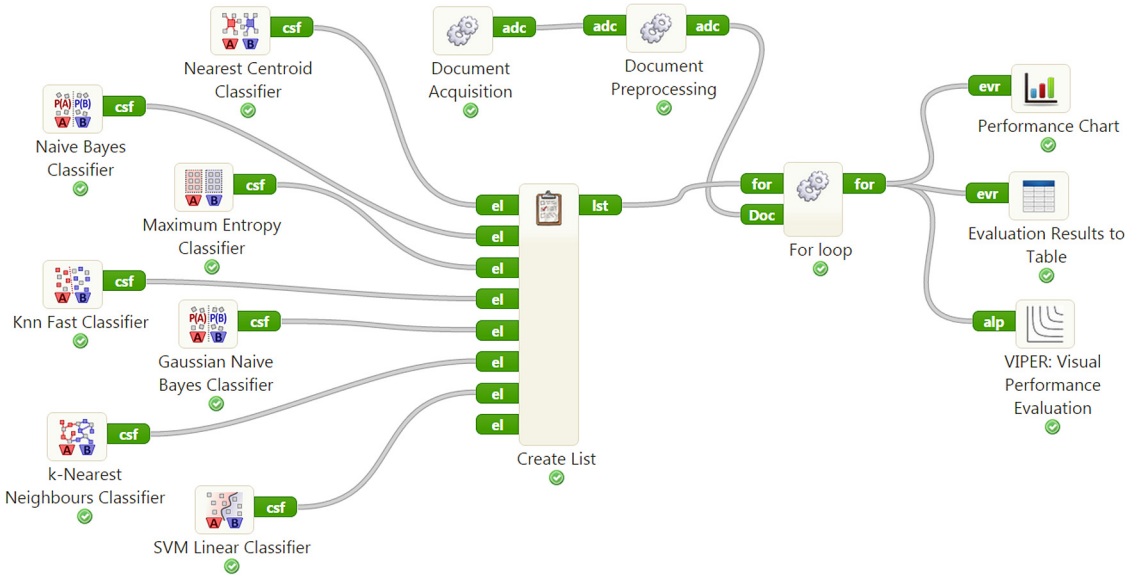

Fig. 5. The workflow for evaluating 7 classifiers from different text mining libraries. The workflow is available at http://textflows.org/workflow/350/.

The two common steps in the three presented use cases are the *Document acquisition* and *Document preprocessing* steps. These two steps were implemented as subprocesses in the main workflows for the three use cases, as shown in Figs. 5, 9 and 11.

In all three workflows, *Document acquisition* is the first step of the methodology and is responsible for, first, loading the locally stored text file (containing the Kenyan elections document corpus), then labeling the documents with appropriate domain labels and finally converting them into the *AnnotatedDocumentCorpus* data object.

Fig. 4 shows the subprocess in the TextFlows platform for the second step in all three presented methodologies: the *Document preprocessing* workflow. As described in Section 4, category specific hubs are used for applying different preprocessing objects to the *AnnotatedDocumentCorpus* data object. The documents are first split into sentences with LATINO's *Max Entropy Sentence Splitter* and then the sentences are split into tokens with LATINO's *Max Entropy Tokenizer*. Some of these tokens are tagged as stop words using the *Stop Word Tagger* with the predefined Snowball list of English stop words. Finally, the *Porter Stemmer* is used for converting tokens into their stems.

5.1. Classifier comparison for text categorization

In this section we propose a workflow for classifier evaluation on the Kenyan elections dataset. In our experiments we compared 7 different classifiers from different text mining libraries. As shown in Fig. 5, we compared 4 classifiers from LATINO (*Nearest Centroid Classifier, Naive Bayes Classifier, Maximum Entropy Classifier, kNN Fast Classifier*) and 3 classifiers implemented in the scikit-learn library (*Gaussian Naive Bayes Classifier, k-Nearest Neighbors Classifier, SVM Linear Classifier*).

Every algorithm was evaluated using 10-fold stratified cross-validation, as shown in Fig. 6. All cross-validations were performed using the same seed in order to ensure the data was equally split for different classifiers. Fig. 7 shows the

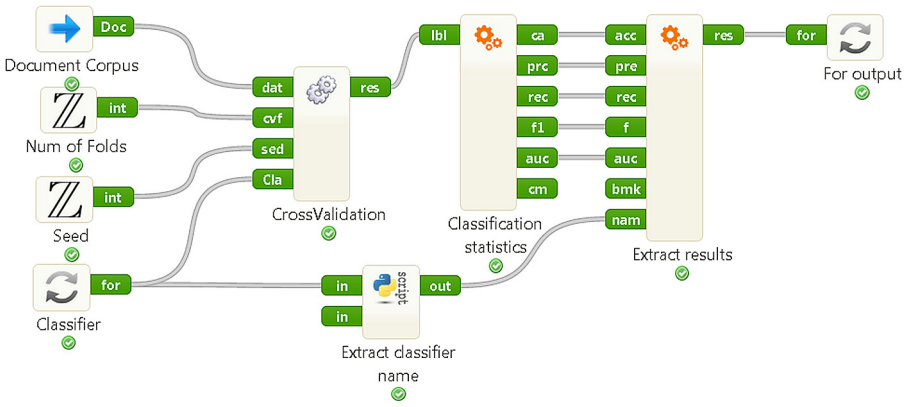

Fig. 6. The *For loop* subprocess which evaluates the input classifier using 10-fold stratified cross-validation (with a constant seed) and extracts the obtained results, which are then visualized as shown in Fig. 5.

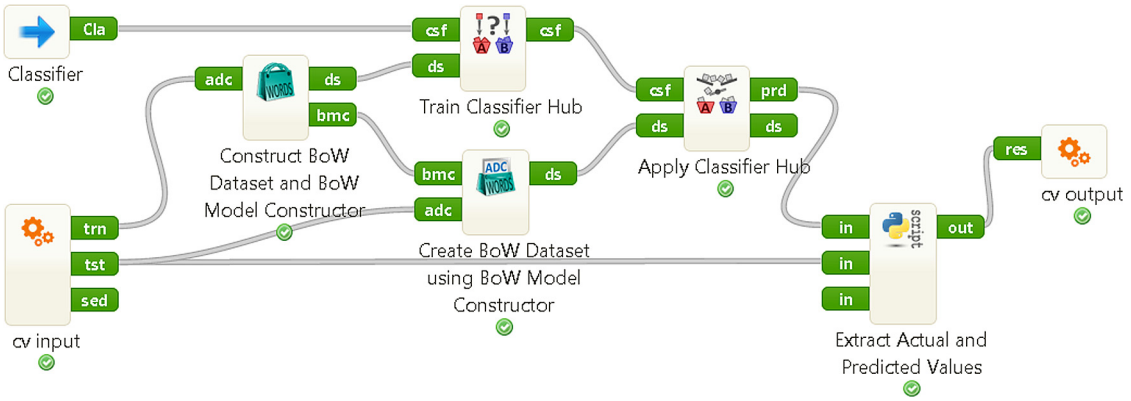

Fig. 7. The subprocess workflow representing the methodology in every fold for the used 10-fold cross-validation. First, the construction of the training and testing sparse matrices is performed. Next, the input classifier model is fitted to the training dataset and used to predict class probabilities for every new document from the input test dataset. The subprocess returns constructed pairs of actual and predicted classes.

methodology behind the *CrossValidation* subprocess which was executed for every cross-validation fold. First, a sparse BoW model dataset and the *BowModelConstructor* instance were generated by the *Construct BoW Dataset and BoW Model Constructor*. We constructed unigrams (n-grams where $n = 1$) out of stemmed values from word tokens, while disregarding stop words in the process. The Bag-of-Words vector space was calculated using the TF-IDF weighting scheme.

The test sparse matrix is required to be constructed using the same parameter settings, IDFs and vocabulary as the training sparse matrix. Therefore, we applied the BoW constructor object (output of the *Construct BoW Dataset and BoW Model Constructor*) to the test ADC data object using the *Create BoW Dataset using the BoW Model Constructor*.

After calculating the training and testing sparse matrices, we fitted the input classifier model to the training dataset using the *Train Classifier Hub*. Next, the *Apply Classifier Hub* received the trained classifier object and returned predicted class probabilities for every new document from the input test dataset. The *Extract Actual and Predicted Values* widget used these probabilities and constructed pairs of actual and predicted classes. These pairs were returned from the *CrossValidation* subprocess and used for calculating different metrics, as shown in Fig. 6.

The results of cross-validation (precision, recall, F-score) were connected to the input of the *VIPER: Visual Performance Evaluation* widget, as shown in Fig. 5, while Fig. 8 presents its visualization of classifier evaluation in the precision-recall plane, where each point represents the result of an algorithm (for the selected target class “Lo”). Points closer to the upper-right corner have higher precision and recall values. F-measure values are presented as isolines (contour lines), which allows a simple comparison of algorithm performance.

Fig. 8 shows that in terms of the F-measure, scikit-learn’s *SVM Linear Classifier* and LATINO’s *Maximum Entropy Classifier* achieved the best results: both algorithms generally achieved a higher percentage of correctly classified examples (higher recall score), and also a slightly higher percentage of correctly classified examples of the target class (higher precision score) compared to other classifiers used. A somewhat lower performance was achieved using LATINO’s *Nearest Centroid Classifier* classifiers, while the k-nearest neighbor and Naive Bayes classifiers performed worse. Detailed results are presented in Table 1.

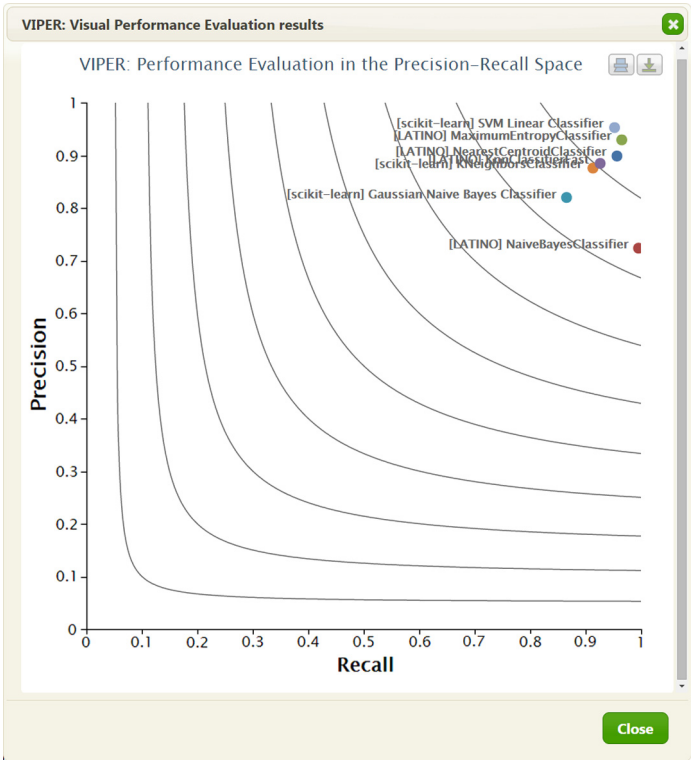

Fig. 8. The VIPER visualization showing evaluation of classifiers from different libraries. This visualization is the result of the workflow presented in Fig. 5.

Table 1
Classifier evaluation on the Kenyan elections dataset.

Library	Classifier	Recall	Prec.	F1 score	Classif. accuracy	AUC
LATINO	Nearest Centroid Classifier	0.96	0.90	0.93	92.42%	0.92
LATINO	Naive Bayes Classifier	1.00	0.72	0.84	80.74%	0.81
LATINO	Maximum Entropy Classifier	0.97	0.93	0.95	94.59%	0.95
LATINO	kNN Fast Classifier	0.93	0.88	0.90	90.26%	0.90
scikit-learn	Gaussian Naive Bayes Classifier	0.87	0.82	0.84	83.77%	0.84
scikit-learn	k-Nearest Neighbors Classifier	0.91	0.88	0.89	89.18%	0.89
scikit-learn	SVM Linear Classifier	0.95	0.95	0.95	95.24%	0.95

The workflow was run within a virtual machine with a dedicated 1 core (Intel i7 2600k) and 4 GB of RAM. The running time for the document acquisition and document preprocessing step is 161 seconds while the comparison of classifiers with the 10 fold cross-validation takes 1486 seconds. The runtime of the entire workflow was 1652 seconds.

5.2. Part-of-speech tagger comparison for text categorization

In this section we describe a workflow for the evaluation of different part-of-speech taggers on the Kenyan elections dataset. In the experiments we compared five different PoS taggers: *English Maximum Entropy PoS Tagger* (from LATINO library) and *PoS Affix Tagger, PoS N-gram Tagger, PoS Brill's rule-based Tagger, PoS Classifier-based Tagger* (from NLTK library). As shown in Fig. 9, PoS tagging widgets from NLTK require tagged sentence data as input, which is used for training the PoS tagger. The TextFlows platform already contains several tagged datasets, which come as a part of the NLTK library, and can be added to workflows through the *NLTK Document Corpus* widget. In this use case we used the annotated Brown corpus.[16]

The training process involves inspecting the tag of each word and storing the most likely tag for any word in a dictionary, which is stored inside the tagger. As it happens, once we have processed several thousand words of English text, most new words will be nouns. In cases when the NLTK PoS tagger is unable to assign a tag from its learned lookup table, it can use a backoff tagger from its input. As shown in Fig. 9 we have set the *PoS Default Tagger* as the backoff tagger for all NLTK

[16] Description of the Brown corpus is available at http://www.nltk.org/book/ch02.html#tab-brown-sources.

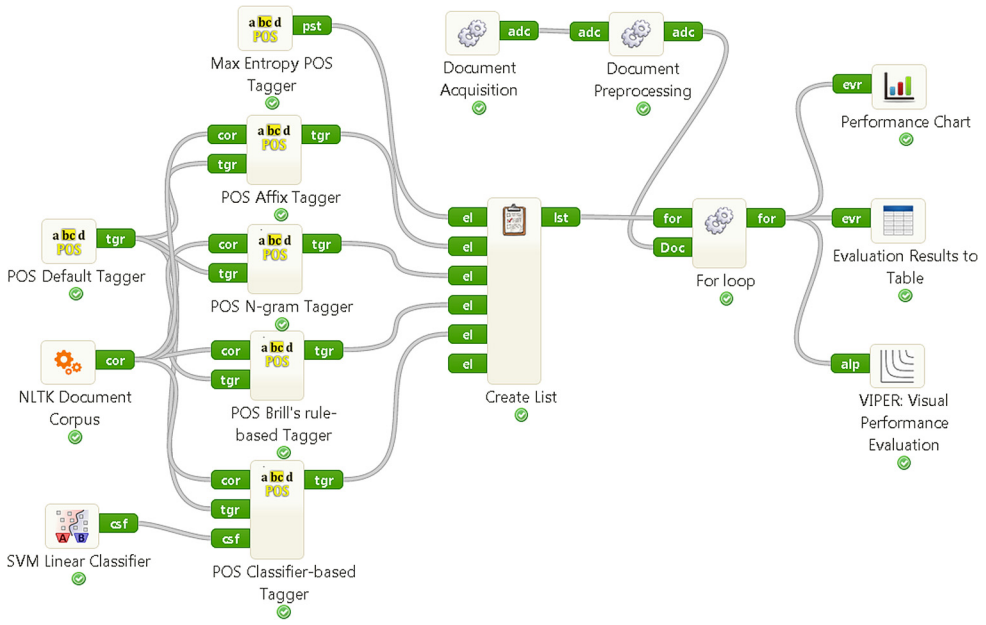

Fig. 9. The workflow used for evaluation of 5 different PoS taggers from various text mining libraries on a text categorization problem. The workflow is available at http://textflows.org/workflow/355/.

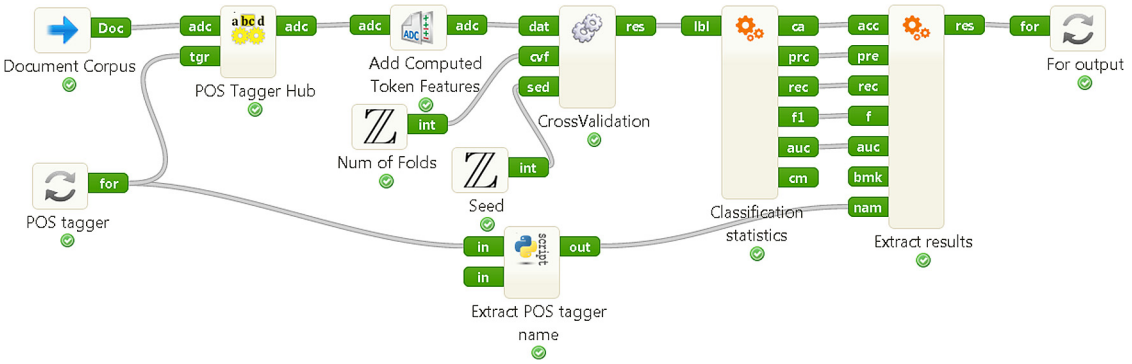

Fig. 10. The subprocess workflow representing the methodology in every fold for the used 10-fold cross-validation. First, the input PoS tagger is applied to the document corpus. Next, the construction of the training and testing sparse matrices is performed. Then, a linear SVM classifier is fitted to the training dataset and used to predict class probabilities for every new document from the input test dataset. Last, the subprocess returns constructed pairs of actual and predicted classes.

PoS taggers. The *PoS Default Tagger* assigns the input tag (e.g., "NN", which is the PoS tag representing a noun) to every single word. Whenever the initial PoS tagger cannot assign a tag to a token it will invoke the input backoff tagger and thus tag the token as a noun. This improves the robustness of the language processing system. The *PoS Classifier-based Tagger* widget also requires input of a classifier, which is learned to predict PoS tags based on the pre-annotated dataset. Every PoS tagger was applied to the document corpus, as shown in Fig. 10. In every iteration (over the list of PoS taggers) of the for loop the input PoS tagger is applied to the tokenized sentences of preprocessed ADC data object using the *PoS Tagger Hub* by generating new features with name "PoS tag" on every elementary (word) token. In order to use PoS tags together with stemmed values, we constructed (for every token) new features named "Stem with PoS" using the *Add Computed Token Features* widget. These features were later used in the CrossValidation subprocess to generate the BoW models. The values of the "Stem with PoS" features were constructed using the *Add Computed Token Features* widget as a combination of stems and PoS tags: "Stem_PoS tag".

Next, 10-fold stratified cross-validation was performed on the generated PoS tagged ADC data objects. Similarly as in the classifier evaluation use case, all cross-validations were performed using the same seed in order to ensure the data was equally split for all PoS taggers. The methodology behind the *CrossValidation* subprocess, which is executed on every cross-validation fold, is similar to the methodology presented in Fig. 7. The only two differences are that cross-validation does not receive a classifier on its input—instead it always uses scikit-learn's linear SVM classifier—and that the *Construct*

Table 2
PoS tagger evaluation on the Kenyan elections database.

Library	Tagger	Recall	Precision	F1 score	Classif. accuracy	AUC
	no PoS tagger	0.98	0.93	0.95	95.24%	0.95
LATINO	Maximum Entropy PoS Tagger	0.98	0.94	0.96	96.10%	0.96
NLTK	PoS Affix Tagger	0.98	0.94	0.96	95.67%	0.96
NLTK	PoS Ngram Tagger	0.98	0.95	0.96	96.10%	0.96
NLTK	PoS Brill Tagger	0.97	0.93	0.95	95.24%	0.95
NLTK+scikit-learn	PoS Classifier Based Tagger (using SVM Linear Classifier)	0.98	0.95	0.96	96.32%	0.96

BoW Dataset and BoW Model Constructor widget uses features constructed by the *Add Computed Token Features* widget instead of stemmed values.

Table 2 shows the results of the presented PoS tagger evaluation workflow. The first row in the table shows the classification results without applying a PoS tagger (see row 3 of Table 1). We see that the usage of a PoS tagger increases the performance of the classifier. The best results were obtained using the NLTK's *PoS Classifier Based Tagger*, which in combination with the LATINO's *Maximum Entropy Classifier* achieved a slightly higher classification accuracy on the Kenyan elections dataset compared to the other PoS taggers.

The experiments were run using the same resources as in the classifier evaluation example—a virtual machine with a setting of 1 core and 4 GB of RAM. The execution time of the entire workflow was 1913 seconds, where 158 seconds were used for document acquisition and preprocessing, while the for loop which compares PoS taggers took 1747 seconds to execute.

5.3. Outlier document detection in categorized document corpora

In this section we propose a workflow for detecting atypical, unusual and/or irregular documents on the Kenyan elections dataset. The idea behind irregularity detection in categorized document corpora is based on early noise filtering approaches by [6], who used a classifier as a tool for detecting noisy instances in data. Noise detection approaches identify irregularities and errors in data and are therefore suitable also for detecting atypical documents in categorized document corpora, which can be considered as outliers of their own document category.

The aim of the NoiseRank (ensemble-based noise detection and ranking) methodology, proposed by [47,45], is to support domain experts in identifying noisy, outlier or erroneous data instances. The user should be able to select the noise detection algorithms to be used in the ensemble-based noise detection process. We have implemented this methodology as a workflow in TextFlows, which now offers widgets implementing classification and saturation noise filters, and enables the inclusion of external user specific noise detection algorithms available as web services. Fig. 11 presents the NoiseRank workflow using the implemented classifiers used for class noise detection.

The NoiseRank methodology workflow returns a visual representation of a list of potential outlier instances, ranked according to the decreasing number of noise detection algorithms which identified an instance as noisy, due to its classification into a class different from its own class label. The ability of NoiseRank to obtain atypical documents was tested on the Kenyan elections corpus. The implemented voting-based irregularity detection method uses four different classifiers acting as noise detection algorithms by identifying misclassified instances.

As in the experiments of Section 5.1 and Section 5.2 we ran the workflow on a virtual machine with 1 CPU core and 4 GB of RAM. The execution time of the entire workflow was 202 seconds, where 148 seconds were used for document acquisition and preprocessing, while the BoW model construction and NoiseRank widget took 52 seconds to execute.

Fig. 12 shows the obtained set of atypical/irregular articles grouped and ranked according to the number of noise detection algorithms that identified them as irregular.

6. Related work

An extensive survey of workflow management platforms, including general data mining platforms—such as RapidMiner [29], Weka [49] and Orange [8]—is out of the scope of this paper, however, we do describe and compare to TextFlows other text mining and natural language processing platforms, starting with a comparison to its predecessor ClowdFlows [28]. In the rest of this section, we concentrate on the key features of the presented platforms: visual programming and execution of scientific workflows, diversity of workflow components, service-oriented architectures, remote workflow execution, big data processing, stream mining, and data sharing. This overview is followed by a comparison of the presented systems with TextFlows.

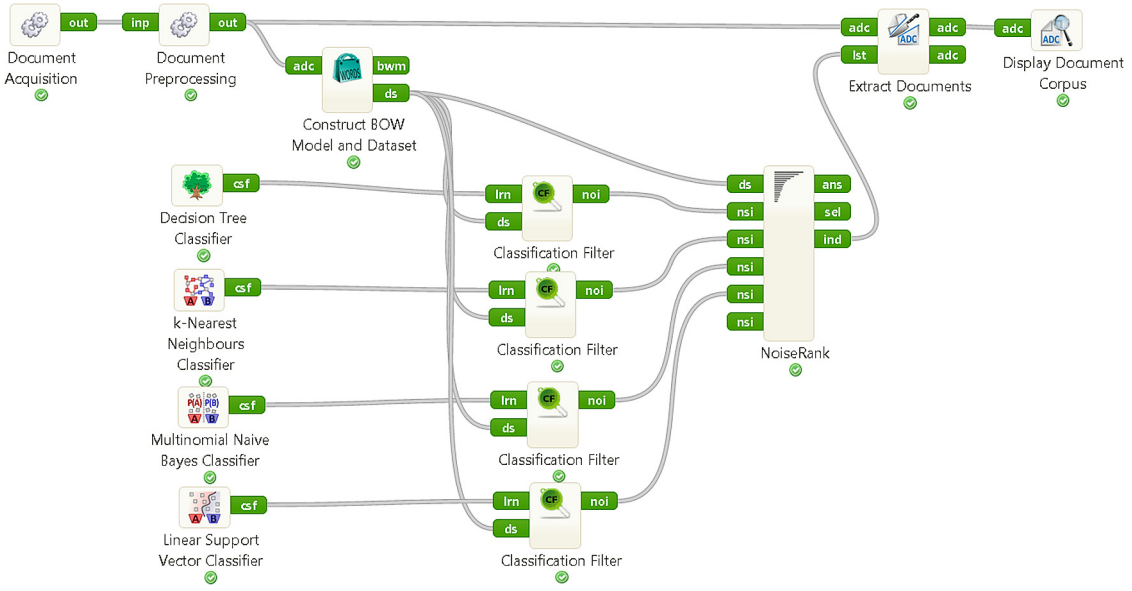

Fig. 11. Example workflow of the NoiseRank methodology, which is available at http://textflows.org/workflow/360/.

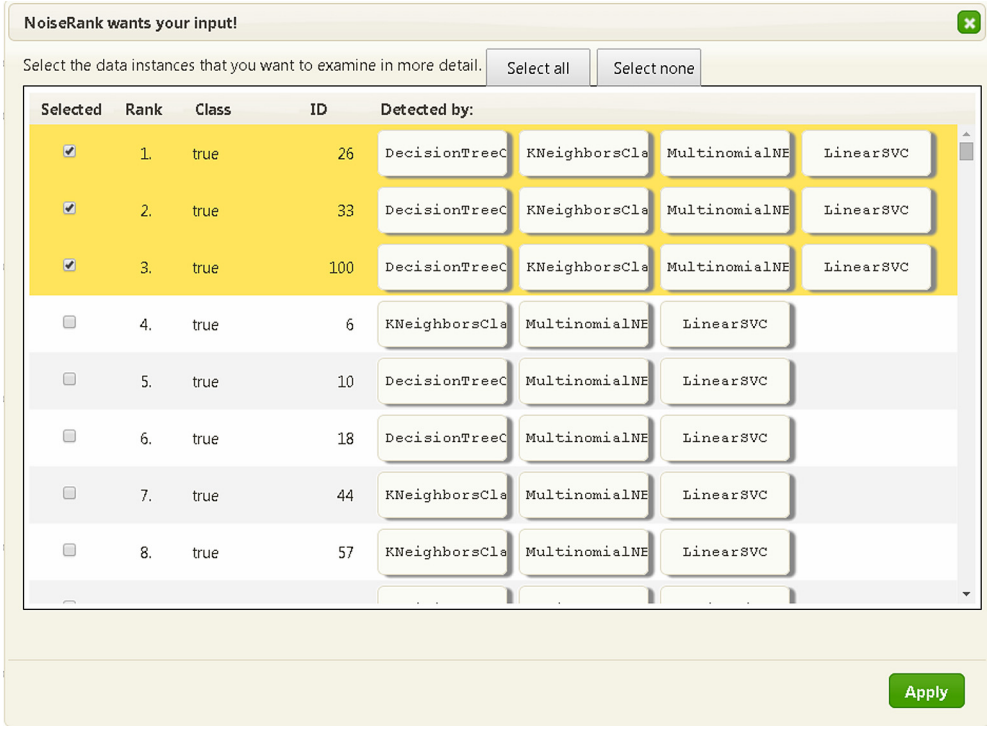

Fig. 12. The NoiseRank interactive widget where the user gets a visual representation of a list of top-ranked potentially noisy instances, which are misclassified according to a decreasing number of elementary noise detection algorithms which identified the instance as noisy. The user can decide which documents he wishes to exclude from the document corpus.

6.1. Comparison with ClowdFlows

TextFlows is a heavily modified fork of the ClowdFlows data mining platform [28], which is a general workflow construction and execution tool for data mining and machine learning. The platform incorporates workflow components for numerous data mining and machine learning projects—including WEKA [49] and Orange [8]—while severely lacking text mining and natural language processing components. Moreover, despite being very useful for workflow construction by the informed developers and end-users, ClowdFlows currently suffers from a somewhat disorganized roster of workflow com-

ponents which may be incompatible with each other. As a result, new users sometimes struggle to construct functioning workflows as there are too many mutually incompatible components.

We created a fork of ClowdFlows in order to maintain cohesion of incorporated text mining and natural language processing workflow components. TextFlows has a completely redesigned roster of workflow components, which are now highly compatible with each other within the platform and easier to use for the expert and novice users alike. In contrast to ClowdFlows, where there are no guidelines for sorting workflow components into categories, TextFlows sorts the components based on the functionality of the components As a result, all widgets that perform similar functions are located in the same category. We have also enriched the user interface with extra information about the workflow components and written the documentation for the provided workflow components and their inputs and outputs.

While we introduced a completely new common text representation structure (cf. Section 2.2), a new widget structure as well as numerous new text mining and new NLP components (cf. Section 4) and workflows (cf. Section 5 for selected examples), the underlying architectures of ClowdFlows and TextFlows remain similar. To summarize, TextFlows is built on top of ClowdFlows, meaning that both the widget execution engine and the core of the ClowdFlows user interface are present in TextFlows. TextFlows still benefits from all security updates, bug fixes and feature upgrades that ClowdFlows receives.

6.2. Survey of workflow management environments for natural language processing

Workflow Management Systems have in the last years become a very hot topic, mostly in the field of bioinformatics and other natural sciences. Lately, however, this trend has also spread to NLP, as evidenced also by recent workshops at the main NLP conferences, e.g., the Workshop "Language Technology Service Platforms: Synergies, Standards, Sharing" at the 9th Language Resources and Evaluation Conference (LREC 2014)[17] and the Workshop on Open Infrastructures and Analysis Frameworks for HLT[18] at the 25th Conference on Computational Linguistics, (COLING 2014).

The current situation with Workflow Management Systems for NLP is very fluid; some well-established systems are slowly starting to be used for NLP applications, while at the same time, new ones are being developed, specifically targeted to NLP. In this section we first overview some of the more important NLP-related Workflow Management Systems, where each platform/project is introduced and its most salient characteristics presented.

6.2.1. Taverna

The set of tools developed by the myGrid[19] team in the U.K. and used primarily for bioinformatics and other life sciences research (having in mind experiment replication) is currently probably the most advanced, richest and easiest to use Workflow Management (Eco)System, and consists of the following main components:

- SoapLab[20] [44] which provides a convenient way to generate web services for command-line software;
- Taverna[21] [16] with its Workflow editor and Server;
- BioCatalogue[22] [2], a registry (for life sciences) where web services can be shared, searched for, annotated with tags, etc.
- myExperiment[23] [41], a social network for sharing, reusing and repurposing public workflows.

As the most important part of the myGrid offerings, we here discuss Taverna, which is conceived as a suite of tools used to design and execute scientific workflows. It combines distributed Web Services and/or local tools into complex analysis pipelines, which can be executed on local desktop machines or through larger infrastructures, such as supercomputers, Grids or cloud environments, using the Taverna Server. The Server allows for workflow execution from web browsers, or through third-party clients; it supports WSDL, REST, GRID and Cloud services, local and distributed command-line scripts, as well as other types of services, such as R-Scripts.

Taverna is connected to myExperiment and BioCatalogue, as well as to other service registries, such as BioVeL,[24] the Biodiversity Virtual e-Laboratory. In addition, Taverna offers an Interaction Service, which enables scientists to select parameters and data during workflow execution, and the Provenance suite, which records service invocations, intermediate and final workflow results.

Taverna workflows can be designed and executed in several ways, to serve different types of workflow users. First, the Taverna Workbench—once downloaded to a local machine—provides an environment for scientists to develop new workflows and test new analysis methods, by either developing workflows from scratch, or by composing them from existing

[17] http://lrec2014.lrec-conf.org/.
[18] http://glicom.upf.edu/OIAF4HLT/.
[19] http://www.mygrid.org.uk/.
[20] http://soaplab.sourceforge.net/soaplab2/.
[21] http://www.taverna.org.uk/.
[22] https://www.biocatalogue.org/.
[23] http://www.myexperiment.org/.
[24] http://www.biovel.eu/.

workflows. Second, workflows can be executed directly through a Taverna Server, which is an environment for serving finished workflows to a larger community of scientists. Here, a single installation of the Server provides access to a collection of workflows, normally through a web interface, called the Taverna Player; in this case, no installation or in-depth knowledge of the workflows is required, but workflows cannot be changed nor can new workflows be added to the collections. The third mode of execution is via a Taverna Lite installation, which provides an intermediate solution, as it allows users not only to run workflows through the web but also to upload new workflows from e.g., myExperiment or other sources. This means that Taverna Lite installations require user authentication, but no local software installation by regular users, as workflow execution also occurs on a server.

The Taverna Workbench, as the most sophisticated means of composing workflows, needs to be first downloaded and installed on a local machine (Windows, Linux or Mac OS X). For third-party services that require a login, Taverna allows credentials to be added at run-time, or to be stored in a purpose-built credential manager. The Workbench allows users to identify and combine services by dragging and dropping them onto the workflow design panel. The services can be from third parties (using e.g., WSDL or REST), but typically also contain local scripts for formatting data and managing service compatibility, known as shim services. Most workflows will need shim services as the analysis services are not usually designed to work together and, therefore, often have incompatible input and output formats. A workflow can also contain nested workflows, so workflows can be components of other workflows. Nested workflows can, for example, control retrieval of data from asynchronous services. Here the nested workflow is executed repeatedly until results are available and the control links between its output and downstream services pause the remainder of the workflow until all preceding results are available. As the workflow runs, the results panel shows progress through the workflow and iterations over data, as well as any errors if there are problems with executions.

The Taverna Server, which executes workflows, can also be downloaded and configured to run with or without login restrictions. It is written in Java, has to be installed on Unix and uses Tomcat with, for secure mode, HTTPS and SSL host certificate. There are various public installations of the Taverna Server, with the best know being the already mentioned BioVeL portal with workflows from the area of biodiversity.

As mentioned, Taverna is currently the most developed and advanced (open source) Workflow Management System, with a host of features and connection capabilities, including fine-grained access management. It is also very popular, e.g., myExperiment currently contains over 2000 Taverna workflows. By far the largest part of the user community is from the field of bioinformatics and other life sciences, where Taverna workflows are typically used in the areas of high-throughput analyses or for evidence gathering methods involving data mining.

Given the power and versatility of Taverna and other myGrid platforms, it is surprising that—apart from a few basic workflows submitted by various individuals—there are few public NLP workflows have been so far implemented in it. In connection with biomedical informatics, there is one published experiment in text mining [25], which has given rise to further research and there is also one platform that makes use of the myGrid building blocks for the purposes of NLP, which is the subject of Section 6.2.2.

6.2.2. PANACEA

In PANACEA[25] [34], a FP7 project that ran 2010–2012, the objective was to automate the stages involved in the acquisition, production, updating and maintenance of language resources required by machine translation systems, and by other applications for processing of natural language.

The architecture of this factory is based on deploying NLP tools as Web Services using SoapLab to generate them for command-line software, this being the standard mode of invocation for most current NLP tools. These individual services can then be combined in the Taverna Workbench and deployed on a Taverna Server.

In the scope of PANACEA various enhancements have been made to the underlying technology, e.g., the possibility to limit in SoapLab the amount of direct data that SOAP messages can transfer; various bugs were also identified and reported to the developers. A common interchange format for the language resources (esp. annotated corpora) was also defined [33], in the first instance XCES [17], because that was previously used by most of the project partners, but in the final version the format moved to the more current Linguistic Annotation Format (LAF) and Graph-based Format for Linguistic Annotations (GrAF) developed in the scope of the ISO/TC37/SC4 technical committee [19]. Finally, the IPR and other legal issues connected to sharing possibly proprietary tools and esp. resources were also considered in PANACEA [1].

The concrete results of the project are made available via ELDA,[26] the European Evaluations and Language resources Distribution Agency, and consist of:

- the PANACEA Registry,[27] currently describing 163 services (not all freely available);
- the PANACEA MyExperiment[28] installation, which allows exploring workflows, but allows executing them only after registration.

[25] http://panacea-lr.eu/.
[26] http://www.elda.org/
[27] http://registry.elda.org/.
[28] http://myexperiment.elda.org/.

The actual web services mostly run on machines (SoapLab or Taverna servers) of the project partners. It should be noted that the ELDA installation is made with an eye to commercializing the platform.

While the PANACEA platform is in place and operational, there does not seem to be any great up-take of this service by the NLP community. An inspection of the PANACEA MyExperiment shows that the platform receives few new workflows or web services, and most of those fairly specific ones by the (ex)project partners.

6.2.3. ARGO

Tsujii lab at the University of Tokyo had a long tradition in combining NLP with biomedical informatics. For example, they were the creators of the GENIA corpus [26], the first and best known biomedical corpus annotated with linguistic information. In connection with their work on NLP for bioinformatics a workflow for text mining U-compare[29] [24], which includes NLP annotation services was developed. U-compare is implemented as an independent Java application, using the Apache UIMA[30] framework. This work was later integrated into Taverna, in the already mentioned work by [25].

Recently, a new workflow management system has been developed on the basis of U-compare, now in the context of the National Centre for Text Mining (NaCTeM) at the University of Manchester, called ARGO[31] [37,38]. ARGO offers the usual array of features, accessed through a browser based interface: the user can upload documents, compose workflows and execute them. A novelty introduced by ARGO is that workflows can have interactive components as well, where the execution of the workflow pauses to receive input from the user. This is useful for workflows which allow for manual annotation of corpora by the user, and ARGO offers several such workflows. However, ARGO does not seem to have any sophisticated utilities for cataloguing available web services or workflows, nor a system of access permissions.

As far as its architecture goes, ARGO continues to be based on UIMA, and uses REST for communication between the components, so other services or workflows can call ARGO defined web services as well. It supports import and export (deserializations and serializations) into RDF, which is the de-facto language of the Semantic Web. The requirement that web services need compatible I/O formats is in ARGO resolved with a powerful system based on cascaded finite-state transducers called Type Mapper [39], which allows for developing needed shim services.

6.2.4. WebLicht

The European Research Infrastructure CLARIN[32] aims to provide the researchers unified single sign-on access to a platform which integrates language-based resources and advanced tools. This is to be implemented by the construction and operation of a shared distributed infrastructure that aims at making language resources, technology and expertise available to the humanities and social sciences research communities. CLARIN is a distributed data infrastructure, with national sites in a number of European countries. These sites provide access to language data repositories, to services and workflows to work with language data, and to expertise that enables researchers to use these resources and tools.

In the scope of national CLARIN portals, various workflows have been developed, with the German WebLicht[33] [15] being the most advanced. While WebLicht is, in some respects, similar to PANACEA, the focus here is not on providing web services for human language technology research, but rather producing annotated corpora for use in linguistic research. Nevertheless, the methods are similar, as on both platforms most web services are devoted to the linguistic annotation of textual input.

WebLicht does not, as does PANACEA, use the myGrid tools and platforms but rather developed its own infrastructure, starting from the wrappers to make the included tools into RESTful web services, to its centralized repository, and the Web editor enabling the composition of the workflows (or toolchains, as they are known in WebLicht, as they are typically limited to one input and one output). As opposed to PANACEA, WebLicht does cover all the standard linguistic annotation steps, for a number of languages, and with often several tools being available for an annotation step. The WebLicht repository includes information about the type of inputs and outputs of individual services, which allows controlling the workflow construction process in the editor to allow connecting only tools that have matching I/O specifications. For example, a part-of-speech tagger needs a tokenizer to be applied to a text before it can be called. As in PANACEA, WebLicht also imposes standards on the I/O interchange format: it uses TCF (Text Corpus Format) [13], a format similar but slimmer than ISO LAF/GrAF, but also provides conversion to LAF/GrAF. With TCF the text and annotations are in one XML file, but with stand-off annotation, with each processing step adding a layer of annotation.

The Web (2.0) based WebLicht editor allows the construction of workflows and their invocation (after a CLARIN(-DE) recognized log-in) and viewing or saving the results. WebLicht has a dedicated viewer, which allows displaying an annotated corpus in tabular format (for tokens and their annotations), lists (for sentences) or as a graphic (for parse trees).

While the WebLicht platform is not open source, the initiative is open to adding new partners who are interested in contributing tools and services. This typically involves wrapping the tools to be contributed to make them RESTful and to take care of I/O requirements, installing this web service on a local machine and then registering them to the WebLicht

[29] http://u-compare.org/.
[30] https://uima.apache.org/.
[31] http://argo.nactem.ac.uk/
[32] http://www.clarin.eu/.
[33] weblicht.sfs.uni-tuebingen.de/.

central repository. Currently, such third-party web services are provided for Finnish, by the University of Helsinki, and Romanian, by their Academy of Sciences.

6.2.5. Language Grid

The Language Grid[34] [20] is a multilingual Internet service platform, developed by Language Grid Project (started in 2006) at the Japanese National Institute of Information and Communications Technology. The Language Grid is based on the service-oriented architecture (SOA), a web-oriented version of the pipeline architecture typically employed by NLP tools. As other Workflow Management Systems it provides three main functions: language service registration and deployment, language service search, and language service composition and execution. Importantly, Language Grid also offers access to a large number of language resources such as online dictionaries and bi-lingual corpora.

In contrast to e.g., myGrid, geared towards running and reproducing scientific experiments, the Language Grid is much more application oriented, with a focus on enabling communication in multilingual communities (via machine translation), the best known example being the support of farming in rural communities in Vietnam, by enabling computer mediated communication between Vietnamese youth and Japanese experts in agriculture. This is also reflected in its architecture, where the users, services and workflows (or "composite services") are centrally administered. And while running such web services is easy with the provided graphical interface, constructing them is more complicated: workflows are composed using WS-BPEL (Web Service Business Process Execution Language) as XML files, rather than in a dedicated Web based or locally installed visual editor. Although Eclipse does provide a visual BPEL editor, which can be used for this process, workflow construction is still more complicated than with e.g., Taverna.

The Language Grid is Open source and the first European node, known as Linguagrid[35] [4] was established in Italy in 2010.

6.2.6. LAPPS Grid

The Language Grid Server also serves as the basis for the U.S. Language Application (LAPPS) Grid project[36] [18]. LAPPS is especially interesting in three aspects. First, it uses advanced standards for data encoding and exchange, in particular the JSON-based serialization for Linked Data (JSON-LD). The JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format that defines a small set of formatting rules for the portable representation of structured data. Because it is based on the W3C Resource Definition Framework (RDF), JSON-LD is simple to map to and from other graph-based formats, in particular the already mentioned ISO LAF/GrAF [19]. It also enables services to reference categories and definitions in web-based repositories and ontologies, such as those of ISOcat.[37]

Second, it uses the Open Advancement approach (developed in the making of IBM's Watson [10]) for component- and application-based evaluation, that has been successful in enabling rapid identification of frequent error categories within modules and documents, thus contributing to more effective investment of resources in both research and application assembly. The LAPPS Grid thus envisions scenarios where it is possible for users to rapidly (re)configure and automatically evaluate a (changed) pipeline on a chosen dataset and metrics.

Third, workflows are planned to incorporate a comprehensive model for addressing constraints on the intellectual property used in the LAPPS Grid, making it maximally open to users ranging from open source developers to commercial users of language services.

6.3. Comparison with TextFlows

The previous sections presented the more widely used Workflow Management Systems, with a focus on those that are also or primarily used for NLP and that support distributed and/or remote processing. So, for example, we do not treat Moa[38] (not to be confused with MOA,[39] an open source framework for data stream mining) and similar systems that are meant to develop workflows on a local server. In this section we compare the overviewed systems, together with TextFlows along several dimensions that affect the usefulness of each system. In the following section we first present an overview table and then discuss the salient dimension.

6.3.1. Open source

The first dimension, summarized in Table 3, concerns the question whether the Workflow Management Systems is open source, i.e., whether it is possible to download the complete system and install it on local server(s). This is important in cases where the system is to be an internal one, not accessible to third parties, e.g., for data privacy protection or where local modifications to the system are desired. In the general case, this option is not really needed, as it is much easier to simply use the official system. Nevertheless, it is desirable to at least have this option available. Of the surveyed systems, Taverna

[34] http://langrid.org/.
[35] http://www.linguagrid.org/.
[36] http://lapps.anc.org/.
[37] http://www.isocat.org/
[38] http://moa.readthedocs.org/.
[39] http://moa.cms.waikato.ac.nz/

Table 3
Comparison of NLP platforms regarding open source.

Taverna	Yes (Java)
PANACEA	No
ARGO	No
WebLicht	No
Language Grid	Yes (Java, PostgreSQL, Tomcat)
TextFlows	Yes (Python)

Table 4
Comparison of NLP platforms regarding workflow sharing.

Taverna	With myExperiment, BioCatalogue, etc.
PANACEA	Own installation of BioCatalogue and MyExperiment
ARGO	Local registry
WebLicht	Local registry
Language Grid	Local registry
TextFlows	Local registry

Table 5
Comparison of NLP platforms regarding simplicity of use.

Taverna	Difficult
PANACEA	Difficult
ARGO	Easy for workflow composition
WebLicht	Easy for workflow composition
Language Grid	Difficult
TextFlows	Easy for workflow composition

is available under OS, and can be installed on all major platforms (Windows, Mac, Linux) with precompiled distributions or in source Java and has few prerequisites; Language Grid (available from SourceForge) also runs on all platforms and requires PostgreSQL and Tomcat, with the set-up being rather complicated. TextFlows is also open source and publicly available with all the prerequisites wrapped in the installation script.

6.3.2. Workflow sharing

All systems offer workflow sharing, as this is the essence of building such systems, however, they differ in whether the platform itself provides the registry and exploration service of registries or whether they make use of associated services for workflow sharing and discovery. As can be seen from Table 4, most systems provide their own sharing platform, with the exception of Taverna and PANACEA, both of which use myExperiment as a social network platform and BioCatalogue (and, in the case of Taverna, other catalogues as well) as a registry for public web services.

6.3.3. Simplicity of use

This dimension, summarized in Table 5 describes how difficult it is to start using a particular system, not so much from the point of view of executing ready-made workflows but of designing workflows and adding new web services to the platforms. Taverna provides a dedicated Workflow Management System, which is, however, mostly due to the variety of options, reportedly quite difficult to master, also because the details of web service communication have to be understood by the user and complex types decomposed into atomic parts. It is also possible to add new web services using SoapLab. The composition of workflows in PANACEA follows that of Taverna, so the same (dis)advantages apply. ARGO provides a Web-based interface, which allows for graphical composition of workflows. It also provides a sophisticated system for adding new web services, including a testing environment. WebLicht supports web-based workflow composition, with the registry constraining which web services can be attached to the workflow, depending on their I/O requirements; however, new web services cannot be added by users. Language Grid does enable composition of new workflows, but this is an off-line and rather complicated process. Finally, TextFlows offers, via a web interface, easy composition of workflows. Adding new web SOAP-based services with WSDL available is also trivial (based on the user-supplied URL of WSDL, new workflow components are created automatically from service's functions).

6.3.4. I/O protocols

This dimension, summarized in Table 6, concerns input/output protocols. The options supported by the communication protocols between the web services in a workflow have mostly to do with the age of the system: the older ones typically prefer WSDL and SOAP, while the newer ones choose the simpler REST.[40]

[40] Note that the WSDL protocol referred to here is version 1.0 or 1.1; WSDL 2.0 offers (limited) support for RESTful web services.

Table 6
Comparison of NLP platforms regarding I/O protocols.

Taverna	WSDL+SOAP, REST
ARGO	REST
PANACEA	WSDL+SOAP
WebLicht	REST
Language Grid	WSDL+SOAP
TextFlows	WSDL+SOAP, REST, JSON-WSP

When dealing with NLP workflows, standards become very important. NLP components in the main perform pipeline processing, where each tool/service adds another layer of annotation to the base text where it typically needs to have access to previous annotations. Furthermore, the input text might itself contain annotations, such as document structure. The wish for interoperability lead to the development of standards for text (corpus) annotation; however there exist a number of such standards and best practices, and different systems use different ones. TAVERNA, not being targeted toward NLP, is standard agnostic, and relies on implementers of workflows to provide the necessary conversion (shiv) services, i.e., conversion routines to the format that the downstream service(s) expect. In the context of NLP it is also possible to develop shiv services that convert not only outputs but also inputs, taking one of the commonly accepted standards as the pivot encoding. This is the route taken by PANACEA, where each web service is wrapped to accept and produce an output: for primary data this is an XCES encoded file, while text annotations use the LAF/GrAF standard with stand-off annotations. ARGO is based on UIMA and does not enforce any NLP standards in its pipelines. However, ARGO did pioneer a new method of aligning I/O requirements of component web services: rather than relying on shiv services, which need programming skills, it supports a Type Mapper, i.e., a dedicated rule-based analytic for transcribing feature structures between types needed for particular web services. It also supports export and type mapping to RDF.

WebLicht uses a local format TCF, which is, however, quite similar to ISO LAF/GrAF and conversion to these is provided. In WebLicht all web services are expected to use TCF, where the conversion is typically implemented as a wrapper around each annotation tool. Furthermore, the WebLicht Registry included information about the prerequisites for each web service and allows only chaining web services that do not violate these constraints. Language Grid has a very different scheme from most other services, and the details of how the interfaces are to be configured are rather hard to come by. In general, it defines an ontology of Web services, which then specifies also their I/O formats. As the focus is on dictionaries and machine translation, it is these kinds of formats that are defined. Finally, TextFlows, as Taverna, does not impose any standards in its I/O formats. Mostly plain text, JSON, XML and, for efficiency, serialized Python data structures and objects (in particular, the *AnnotatedDocumentCorpus* instances) are exchanged between workflow components.

7. Conclusions and directions for future work

This paper presented TextFlows, an open source platform featuring workflow components for text mining and natural language processing. TextFlows provides a common graphical user interface and joins several text mining, visualization and machine learning libraries under a single unifying platform, expanding their usability for researchers, practitioners and non-experts. The architecture of the platform is scalable, allows for many users and workflow and dataset sharing.

The usability of the platform was demonstrated on three illustrative use cases, showing that components developed for different systems and in different programming languages can be run within a single coherent system and may be represented as a visually constructed workflow available on the web. The ease of sharing the completed workflows and results over the web allows for TextFlows to become a central integration platform for many text mining and natural language processing tasks. We have compared our work to related platforms and shown the differences, benefits, and drawback of using TextFlows instead of other text mining platforms.

There are several directions for future work. We plan to expand the TextFlows widget repository with additional text preprocessing components (such as chunking, term extraction, syntactic parsing), extend the batch of weighting schemes for generating the BoW models, and include various algorithms for clustering. We also plan to extend the literature-based discovery package to include more widgets for cross-domain bridging term discovery. Furthermore, we will connect the platform to various external tools to better assist the user in the process of exploration and visualization of results, such as TopicCircle [22] for cluster visualization.

Second, by using public data on user workflows, submitted to the public version of the TextFlows platform, we will construct a recommender system based on the data on previously executed workflows that will enable computer-assisted construction of text mining workflows and bring the platform even closer to non-experts in terms of usability.

A current weakness of TextFlows is the inability to deal with very large data. As part of future work we plan to integrate the stream mining and big data mining features of ClowdFlows to the TextFlows platform. This will allow performing natural language processing on a larger scale, as well as preforming different tasks on streams of data such as website news feeds, or real time data from social networks such as Twitter and Facebook.

Finally, we wish to simplify the installation procedures of TextFlows to private servers by providing one-click deployment to services such as Amazon Elastic Compute Cloud (EC2) and Microsoft Azure.

We have released the sources of the TextFlows platform under an open source MIT license, available at https://github.com/xflows/textflows. Detailed deployment instructions are provided with the source code. A public installation of the platform is available for online use at http://textflows.org.

Acknowledgements

We are grateful to Borut Sluban and Senja Pollak for their previous work on the Kenyan elections document corpus. This work was supported by the Slovenian Research Agency and the FP7 European Commission FET projects MUSE (grant no. 296703) and ConCreTe (grant no. 611733).

References

[1] V. Arranz, O. Hamon, On the way to a legal sharing of web applications in NLP, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, Istanbul, Turkey, European Language Resources Association (ELRA), 2012, pp. 2965–2970.
[2] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, C.A. Goble, BioCatalogue: a universal catalogue of web services for the life sciences, Nucleic Acids Res. 38 (Web-Server-Issue) (2010) 689–694.
[3] S. Bird, NLTK: the natural language toolkit, in: Proceedings of the COLING/ACL on Interactive Presentation Sessions, Association for Computational Linguistics, 2006, pp. 69–72.
[4] A. Bosca, L. Dini, M. Kouylekov, M. Trevisan, Linguagrid: a network of linguistic and semantic services for the Italian language, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, Istanbul, Turkey, European Language Resources Association (ELRA), 2012, pp. 3304–3307.
[5] E. Brill, A simple rule-based part of speech tagger, in: Proceedings of the Workshop on Speech and Natural Language, Association for Computational Linguistics, 1992, pp. 112–116.
[6] C.E. Brodley, M.A. Friedl, Identifying mislabeled training data, J. Artif. Intell. Res. 11 (1999) 131–167.
[7] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth, CRISP-DM 1.0 step-by-step data mining guide, http://www.crisp-dm.org/CRISPWP-0800.pdf, 2000.
[8] J. Demšar, B. Zupan, G. Leban, T. Curk, Orange: from experimental machine learning to interactive data mining, in: Proceedings of Knowledge Discovery in Databases, PKDD, 2004, pp. 537–539.
[9] R. Feldman, J. Sanger, The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data, Cambridge University Press, 2007.
[10] D.A. Ferrucci, E.W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J.M. Prager, N. Schlaefer, C.A. Welty, Building Watson: an overview of the DeepQA project, AI Mag. 31 (3) (2010) 59–79.
[11] M. Grčar, Mining text-enriched heterogeneous information networks, Ph.D. thesis, Jožef Stefan International Postgraduate School, 2015.
[12] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, USA, 2006.
[13] U. Heid, H. Schmid, K. Eckart, E. Hinrichs, A corpus representation format for linguistic web services: the D-SPIN text corpus format and its relationship with ISO standards, in: Proceedings of the Seventh International Conference on Language Resources and Evaluation, LREC'10, Valletta, Malta, European Language Resources Association (ELRA), 2010, pp. 494–499.
[14] D. Hiemstra, A probabilistic justification for using TF-IDF term weighting in information retrieval, Int. J. Digit. Libr. 3 (2) (2000) 131–139.
[15] M. Hinrichs, T. Zastrow, E. Hinrichs, WebLicht: web-based LRT services in a distributed eScience infrastructure, in: Proceedings of the Seventh International Conference on Language Resources and Evaluation, LREC'10, Valletta, Malta, European Language Resources Association (ELRA), 2010.
[16] D. Hull, K. Wolstencroft, R. Stevens, C.A. Goble, M.R. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services, Nucleic Acids Res. 34 (Web-Server-Issue) (2006) 729–732.
[17] N. Ide, P. Bonhomme, L. Romary, XCES: an XML-based encoding standard for linguistic corpora, in: Proceedings of the Second International Language Resources and Evaluation Conference, Paris, European Language Resources Association, 2000, pp. 825–830.
[18] N. Ide, J. Pustejovsky, C. Cieri, E. Nyberg, D. Wang, K. Suderman, M. Verhagen, J. Wright, The language application grid, in: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC'14, Reykjavik, Iceland, European Language Resources Association (ELRA), 2014, pp. 22–30.
[19] N. Ide, K. Suderman, GrAF: a graph-based format for linguistic annotations, in: Proceedings of the Linguistic Annotation Workshop, LAW '07, Stroudsburg, PA, USA, Association for Computational Linguistics, 2007, pp. 1–8.
[20] T. Ishida, The Language Grid: Service-Oriented Collective Intelligence for Language Resource Interoperability, Springer Science & Business Media, 2011.
[21] M. Juršič, B. Cestnik, T. Urbančič, N. Lavrač, Cross-domain literature mining: finding bridging concepts with CrossBee, in: Proceedings of the 3rd International Conference on Computational Creativity, 2012, pp. 33–40.
[22] M. Juršič, B. Cestnik, T. Urbančič, N. Lavrač, HCI empowered literature mining for cross-domain knowledge discovery, in: Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data, Springer, 2013, pp. 124–135.
[23] M. Juršič, I. Mozetič, T. Erjavec, N. Lavrač, Lemmagen: multilingual lemmatisation with induced ripple-down rules, J. Univers. Comput. Sci. 16 (9) (2010) 1190–1214.
[24] Y. Kano, W. Baumgartner, L. McCrohon, S. Ananiadou, K. Cohen, L. Hunter, J. Tsujii, U-Compare: share and compare text mining tools with UIMA, Bioinformatics 25 (15) (2009) 1997–1998.
[25] Y. Kano, P. Dobson, M. Nakanishi, J. Tsujii, S. Ananiadou, Text mining meets workflow: linking U-Compare with Taverna, Bioinformatics 26 (19) (2010) 2486–2487.
[26] J.-D. Kim, T. Ohta, Y. Tateisi, J. Tsujii, GENIA corpus – a semantically annotated corpus for bio-textmining, in: ISMB (Supplement of Bioinformatics), 2003, pp. 180–182.
[27] C. Kohlschütter, P. Fankhauser, W. Nejdl, Boilerplate detection using shallow text features, in: Proceedings of the Third ACM International Conference on Web Search and Data Mining, ACM, 2010, pp. 441–450.
[28] J. Kranjc, V. Podpečan, N. Lavrač, ClowdFlows: a cloud based scientific workflow platform, in: P.A. Flach, T.D. Bie, N. Cristianini (Eds.), Proceedings of Machine Learning and Knowledge Discovery in Databases, ECML/PKDD (2), Springer, 2012, pp. 816–819.
[29] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler, Yale: rapid prototyping for complex data mining tasks, in: L. Ungar, M. Craven, D. Gunopulos, T. Eliassi-Rad (Eds.), Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, 2006, pp. 935–940.
[30] A.Y. Ng, Feature selection, L1 vs. L2 regularization, and rotational invariance, in: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, 2004, p. 78.
[31] G. Paltoglou, M. Thelwall, A study of information retrieval weighting schemes for sentiment analysis, in: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2010, pp. 1386–1395.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[33] M. Poch, N. Bel, Interoperability and technology for a language resources factory, in: Proceedings of the Workshop on Language Resources, Technology and Services in the Sharing Paradigm, Chiang Mai, Thailand, Asian Federation of Natural Language Processing, 2011, pp. 32–40.
[34] M. Poch, A. Toral, O. Hamon, V. Quochi, N. Bel, Towards a user-friendly platform for building language resources based on web services, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, Istanbul, Turkey, European Language Resources Association (ELRA), 2012, pp. 1156–1163.
[35] S. Pollak, Text classification of articles on Kenyan elections, in: Proceedings of the 4th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics, 2009, pp. 229–233.
[36] S. Pollak, R. Coesemans, W. Daelemans, N. Lavrač, Detecting contrast patterns in newspaper articles by combining discourse analysis and text mining, Pragmatics 21 (4) (2013) 674–683.
[37] R. Rak, A. Rowley, W. Black, S. Ananiadou, Argo: an integrative, interactive, text mining-based workbench supporting curation, Database, J. Biol. Databases Curation (2012), http://database.oxfordjournals.org/content/2012/bas010.full.
[38] R. Rak, A. Rowley, J. Carter, S. Ananiadou, Development and analysis of NLP pipelines in Argo, in: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Association for Computational Linguistics, 2013, pp. 115–120.
[39] R. Rak, A. Rowley, J. Carter, R.T.B. Batista-Navarro, S. Ananiadou, Interoperability and customisation of annotation schemata in Argo, in: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC'14, European Language Resources Association (ELRA), 2014, pp. 3837–3842.
[40] P. Refaeilzadeh, L. Tang, H. Liu, Cross-validation, in: Encyclopedia of Database Systems, Springer, 2009, pp. 532–538.
[41] D.D. Roure, C. Goble, R. Stevens, The design and realisation of the myExperiment virtual research environment for social sharing of workflows, Future Gener. Comput. Syst. 25 (2009) 561–567.
[42] G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, Inf. Process. Manag. 24 (5) (1988) 513–523.
[43] S. Scott, S. Matwin, Feature engineering for text classification, in: Proceedings of 16th International Conference on Machine Learning, ICML'99, 1999, pp. 379–388.
[44] M. Senger, P. Rice, T. Oinn, Soaplab: a unified sesame door to analysis tools, in: UK e-Science All Hands Meeting, National e-Science Centre, 2003, pp. 509–513.
[45] B. Sluban, D. Gamberger, N. Lavrač, Ensemble-based noise detection: noise ranking and visual performance evaluation, Data Min. Knowl. Discov. 28 (2) (2014) 265–303.
[46] B. Sluban, N. Lavrač, ViperCharts: visual performance evaluation platform, in: Proceedings of Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 650–653.
[47] B. Sluban, S. Pollak, R. Coesemans, N. Lavrač, Irregularity detection in categorized document corpora, in: Proceedings of the Eight International Conference on Language Resources and Evaluation, LREC'12, 2012, pp. 1598–1603.
[48] N. Smalheiser, D. Swanson, Using ARROWSMITH: a computer-assisted approach to formulating and assessing scientific hypotheses, Comput. Methods Programs Biomed. 57 (3) (1998) 149–154.
[49] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd ed., Morgan Kaufmann, Amsterdam, 2011.