Internal Report IJS-DP-12226

Deliverable 2.1: Upgrading the ClowdFlows browser-based workflow management platform

Jožef Stefan Institute Version 0.6 FINAL

Document administrative information	
Project acronym:	HinLife
Project number:	J7-7303
Deliverable number:	D2.1
Deliverable full title:	Upgrading the ClowdFlows browser-based workflow management platform
Document identifier:	HinLife-del-D2.1-ClowdFlows-v0.6.docx
Lead partner short name:	Jožef Stefan Institute
Report version:	0.6, Final
Report preparation date:	22/11/2016
Lead author:	Dragana Miljković
Co-authors:	Anže Vavpetič, Nada Lavrač, Vid Podpečan, Janez Kranjc, Bojan Cestnik
Status:	Final

Introduction

Within work package WP2 (Platform development and benchmarking) of the HinLife project we continued developing and upgrading the web-based ClowdFlows platform for visual design, execution and monitoring of data mining and knowledge discovery workflows. Visual programming simplifies the construction of complex workflows by providing basic building blocks (widgets) which can be connected into a workflow in a web browser. It will also support repeatability of experiments by saving constructed workflows and parameters, and provide an intuitive structuring of complex parts of workflows by introducing the notion of meta-workflow (workflow of workflows). The platform became suitable for non-experts due to the representation of complex procedures as sequences of simple processing steps. The results of this task are the versions of the prototype software tool with increasing functionality to allow realizing the use case scenarios. This web application enables simple composition, execution and testing of software modules as well as simple comparison of results in benchmarking tasks. Results of our ClowdFlows development work were published in journal papers [1] [2].

ClowdFlows development until 2016

ClowdFlows is a cloud-based web application that can be accessed and controlled from anywhere using a web browser, while the processing is performed in a cloud of computing nodes. The architecture of the ClowdFlows platform is shown in Figure 1. The platform currently consists of the following components: a graphical user interface, a core processing server, a database, a stream mining daemon, a broker that delegates execution tasks, distributed workers, web services, and a module for big data analysis in batch mode.

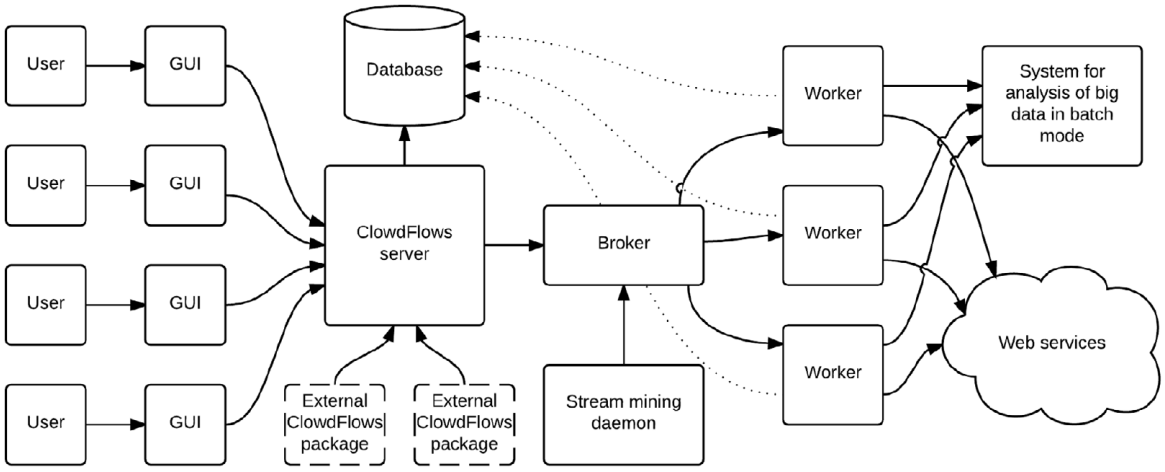

Figure 1. An overview of the ClowdFlows platform architecture.

The integral part of the ClowdFlows platform is the data model which consists of an abstract representation of workflows and widgets. Workflows are executable graphical representations of complex procedures. A workflow in ClowdFlows is a set of widgets and connections. A widget is a single workflow processing unit with inputs, outputs and parameters. Each widget performs a task considering its inputs and parameters, and then stores the results of the task on its outputs. Connections are used to transfer data between two widgets and may exist only between an output of a widget and an input of another widget. Data is transferred between connections, so each input can only receive data from a connected output. Parameters are similar to inputs, but need to be entered manually by users. Inputs can be transformed into parameters and vice-versa, depending on the users' needs. All the data from the data model are stored in the database.

Use cases

In this section we demonstrate here the use of the platform with two illustrative use cases. The goal of these use cases is to present a few basic features of ClowdFlows.

As a first use case, we have developed a workflow for evaluating and comparing several machine learning algorithm implementations. Decision tree, Naive Bayes and Support Vector Machines algorithms from Weka and Orange are evaluated with the leave-one-out cross-validation method on several publicly available data sets from the UCI repository [3] and the results of the evaluation are presented using the VIPER (Visual Performance Evaluation) [4] interactive performance evaluation charts. The interactive workflow demonstrates the use of web services as workflow components, as the employed Weka algorithms have been made available as web services. The sample workflow for the evaluation and comparison of several machine learning algorithm implementations is shown in Figure 2. This workflow is publicly available at http://clowdflows.org/workflow/6038/. The workflow performs as follows. First, instances of the selected machine learning algorithm implementations are created from the libraries (Weka and Orange) and concatenated into a list. Second, a UCI data set is selected, loaded, and transformed into two different data structures, one for each library. Validation is performed using the leave-one-out method on three pairs of algorithm implementations from different libraries. Confusion matrices are computed, and the results are prepared for visualization. In the final step, the VIPER chart (Visual performance evaluation) is shown. The chart offers interactive visualization of algorithm results in the precision–recall space thus allowing a visual comparison of several algorithms and export of publication quality figures. This performance visualization is shown in Figure 3. The public ClowdFlows installation features many other example workflows including workflows that demonstrate regression[1] and clustering[2].

[1] http://clowdflows.org/workflow/7539/.
[2] http://clowdflows.org/workflow/7492/.

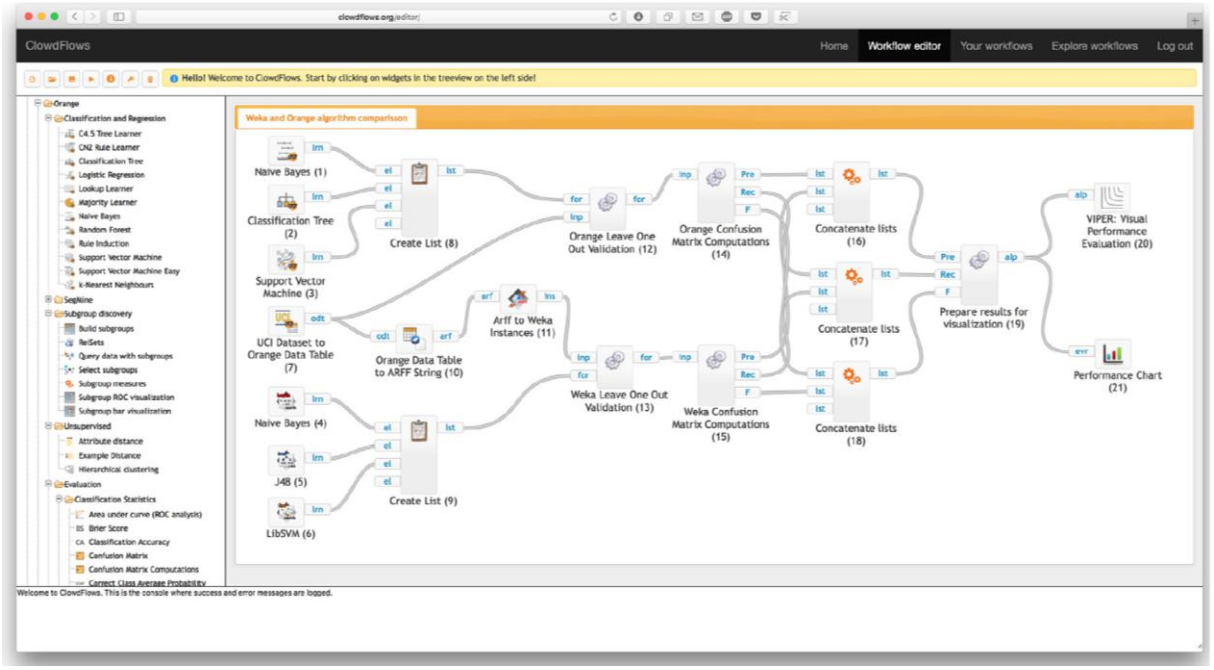

Figure 2. A ClowdFlows workflow for comparison of algorithms from two different machine learning libraries (Weka and Orange). Algorithms are evaluated on a UCI data set using the leave one out cross validation and their performance is visualized using VIPER charts. This workflow is publicly available at http://clowdflows.org/workflow/6038/.

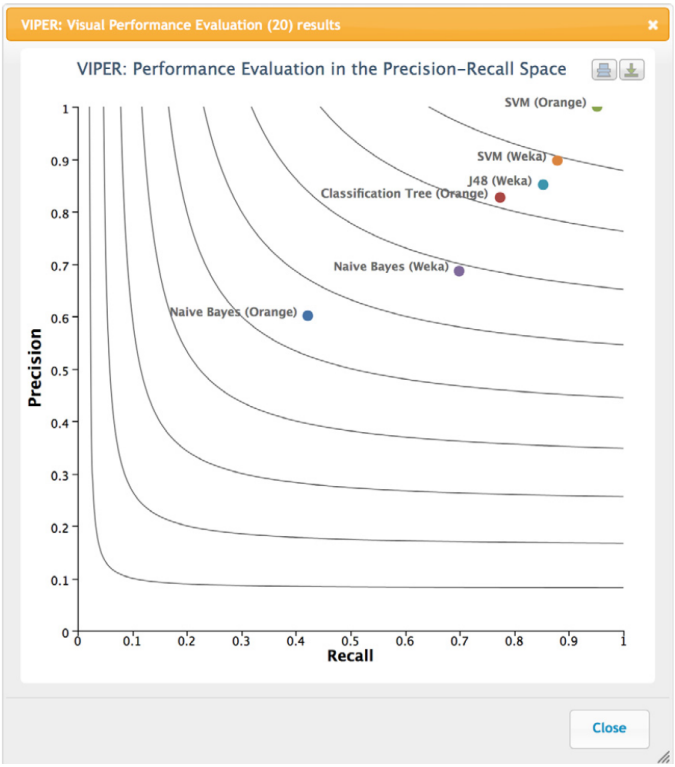

Figure 3. Visual performance evaluation of several machine learning algorithms implemented in ClowdFlows.

The aim of the second use case was to find interesting subsets of patients from the breast cancer dataset. Subgroup discovery (SD) methods can be used to find interesting subsets of objects of a given class. While subgroup describing rules are themselves good explanations of the subgroups, domain ontologies can provide additional descriptions to data and alternative explanations of the constructed rules. Such explanations in terms of higher level ontology concepts have the potential of providing new insights into the domain of investigation. We used this approach to explain subgroups through ontologies on a gene expression profiling use case where groups of patients, identified through SD in terms of gene expression, were further explained through concepts from the Gene Ontology [5] and KEGG orthology [6]. This methodology was implemented as a workflow within the ClowdFlows platform (Figure 4).

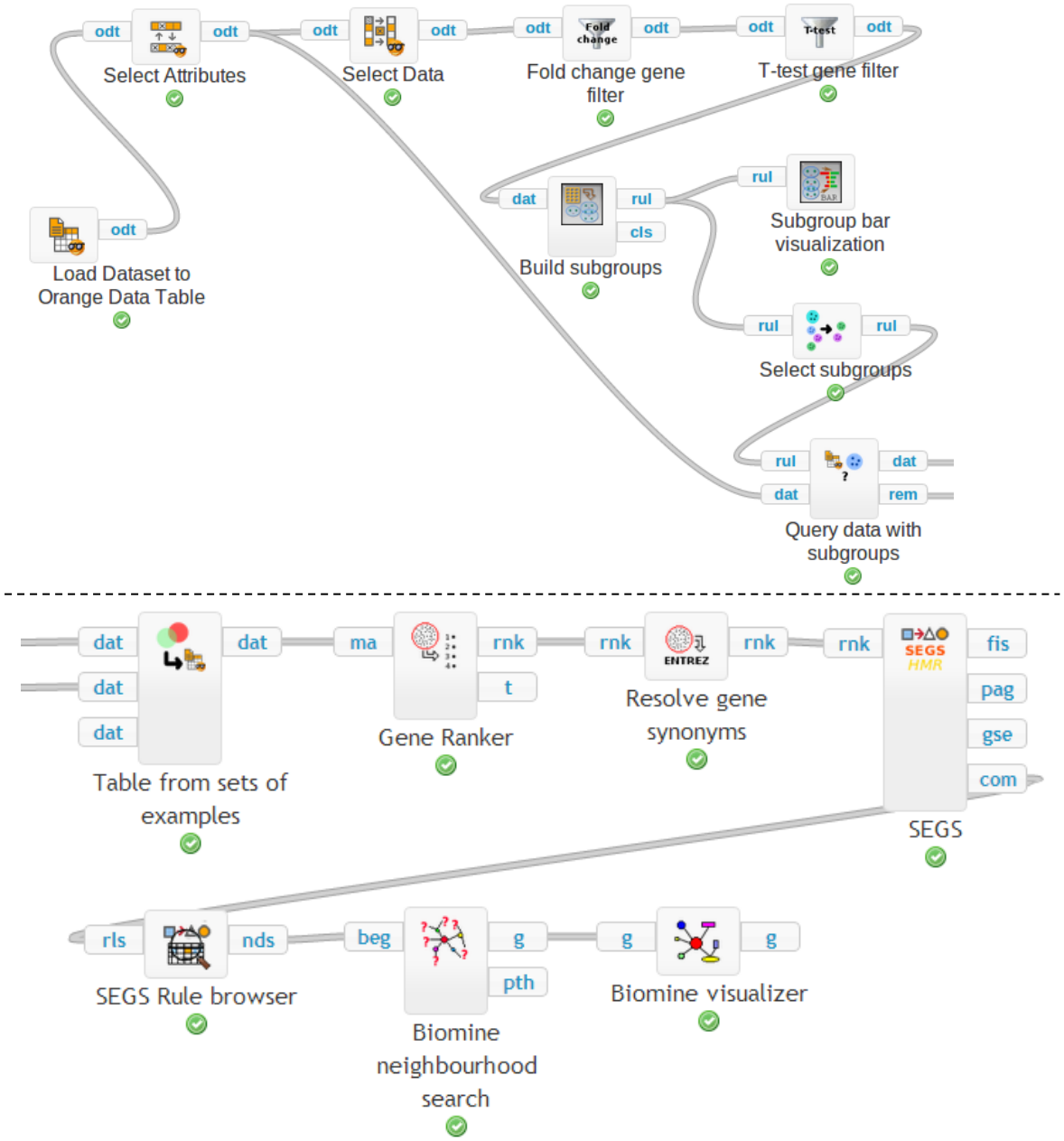

Figure 4. An approach to explaining subgroups through ontologies implemented as workflow in ClowdFlows. The workflow was split into two parts in order to be more easily readable and can be found at http://clowdflows.org/workflow/911/.

TextFlows development in 2016

In 2016 we finalized the implementation of TextFlows, a fork of ClowdFlows, which is a specialized visual programming platform for text mining and natural language processing. TextFlows is 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.

This report presents some of the implemented text mining and language processing modules, and describes some pre-composed workflows. TextFlows is publicly available[3] and its source code[4] is also publicly available 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.

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). 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.

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.

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.).

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. 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.

Use case relevant to HinLife

We demonstrate the use of the platform with a use case relevant to the HinLife project. The workflow shown in Figure 5 consists of seven steps implemented as sub-processes. The connections between

[3] http://textflows.org
[4] https://github.com/xflows/textflows

subprocesses represent the flow of documents from one subprocess to another. Each subprocess is further presented and explained in Figure 6 and Figure , respectively.

In the first three steps the outlier documents are identified and extracted using the NoiseRank outlier detection approach as implemented in TextFlows. The goal of this phase is to extract a set of outlier documents from the whole corpus of input documents. Consequently, by decreasing the size of the input set of documents the second phase becomes more focused, efficient and effective.

In the last four steps components of cross-context bridging term exploration engine CrossBee are executed to facilitate expert-guided bridging term (b-term) analysis. Here, the goal is to further prepare the input documents for b-term visualization and exploration.

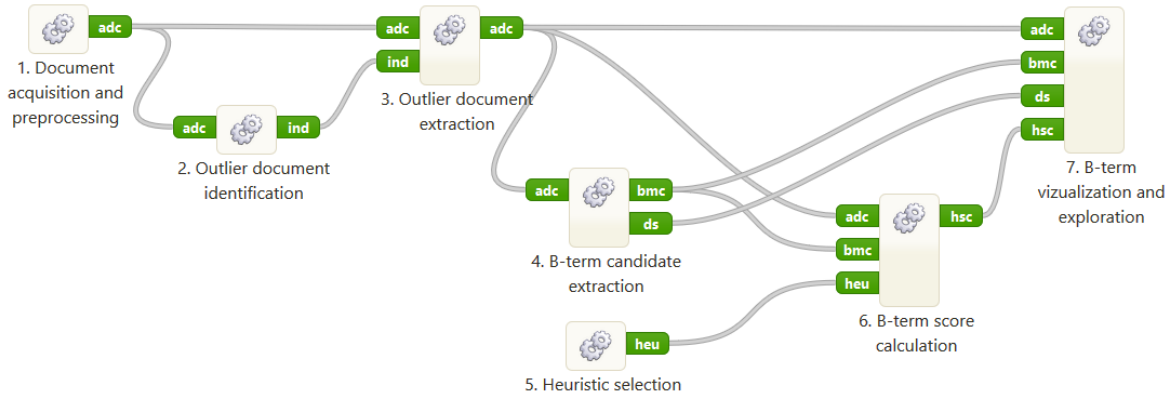

Figure 5. A top-level workflow of the proposed methodology in TextFlows.

Figure 6 presents the detailed workflows for subprocesses 1. Document acquisition and preprocessing, 2. Outlier document identification, and 3. Outlier document extraction.

The main task of the first subprocess (1. Document acquisition and preprocessing) is to read the input set of documents from a file and to transform unstructured text documents into a predefined well-structured data representation containing a set of relevant features for further processing. Each document is represented by a BoW vector of numerical values, one for each feature of the selected representational model. In TextFlows, the preprocessing techniques are based on standard text mining concepts and include Tokenization, Stop words tagging, and Stem/Lemma tagging, which are all present in the workflow of the first subprocess.

The purpose of the two workflows of the second and third subprocesses of Figure 6 is to identify and extract outlier documents from the initial document corpus. The NoiseRank component implements a noise detection strategy in which classifiers are used to detect atypical documents in categorized document corpora, which can be considered as outliers of their own document category.

The main purpose of the NoiseRank component as implemented in TextFlows (widget 2.10. in Figure 6) is to support domain experts in identifying noisy, outlier or erroneous data instances. The users are able to select the noise detection algorithms to be used in the ensemble-based noise detection process. The NoiseRank methodology workflow returns a visual representation of a list of potential outlier documents, ranked according to the decreasing number of noise detection algorithms which identified a document as outlier. So, in addition, the users can obtain a visual representation of top-ranked outlier documents as shown in Figure 8. The feature allows also for manual intervention (inclusion or exclusion from the list of outliers). Note that in our experiments we have chosen the default select-all option.

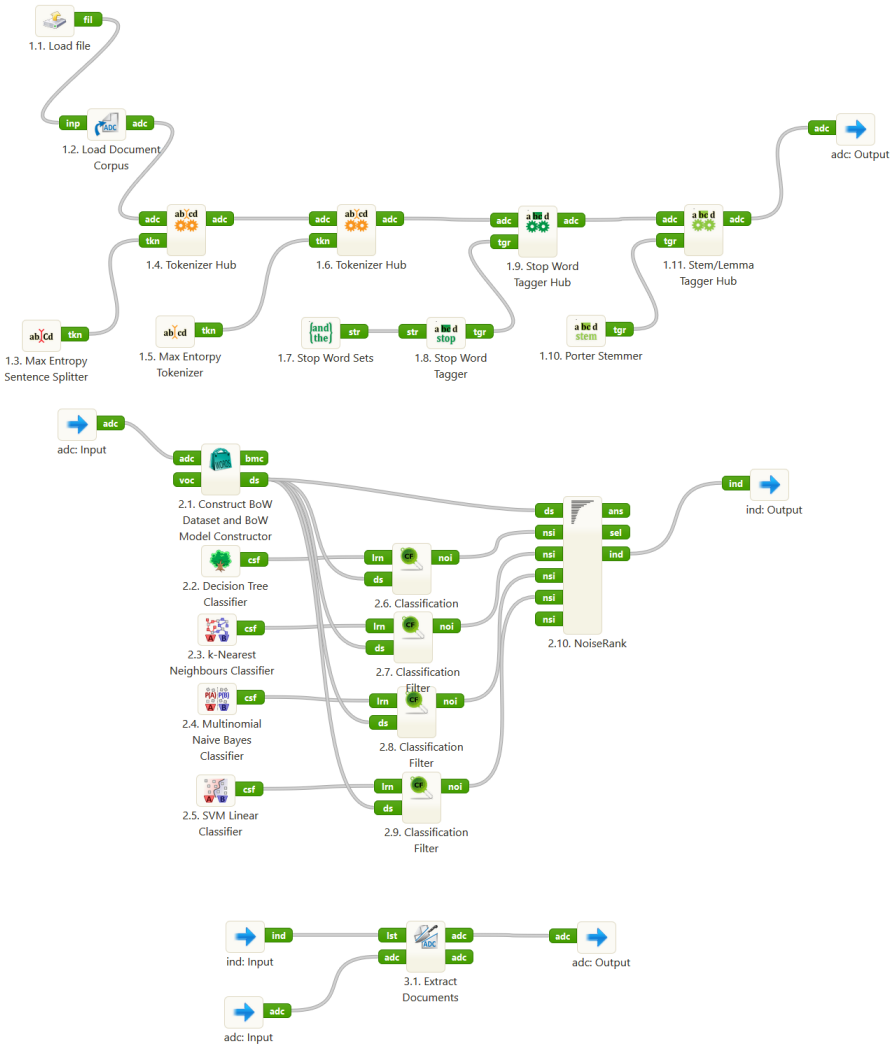

Figure 6. Detailed workflows for the first three sub-processes that are used to identify and extract outlier documents.

Figure 7 presents detailed workflows for subprocesses 4. B-term candidate extraction, 5. Heuristic selection, 6. B-term score calculation and 7. B-term visualization and exploration.

The component 4.1 first constructs BoW representation of the outlier documents. Then, in subprocess 5. the heuristics used to evaluate b-term potential are selected. Subprocess 6. consists of a single widget labeled 6.1. Calculate Term Heuristic Scores. This widget takes as an input several heuristics specifications and performs the actual calculations of the heuristics on each and every b-term. 7.4. Explore in CrossBee widget, which exports the final ranking results and the annotated document corpus into web application CrossBee, is the most important part of subprocess 7. This component enables manual exploration of potential b-terms and respective documents.

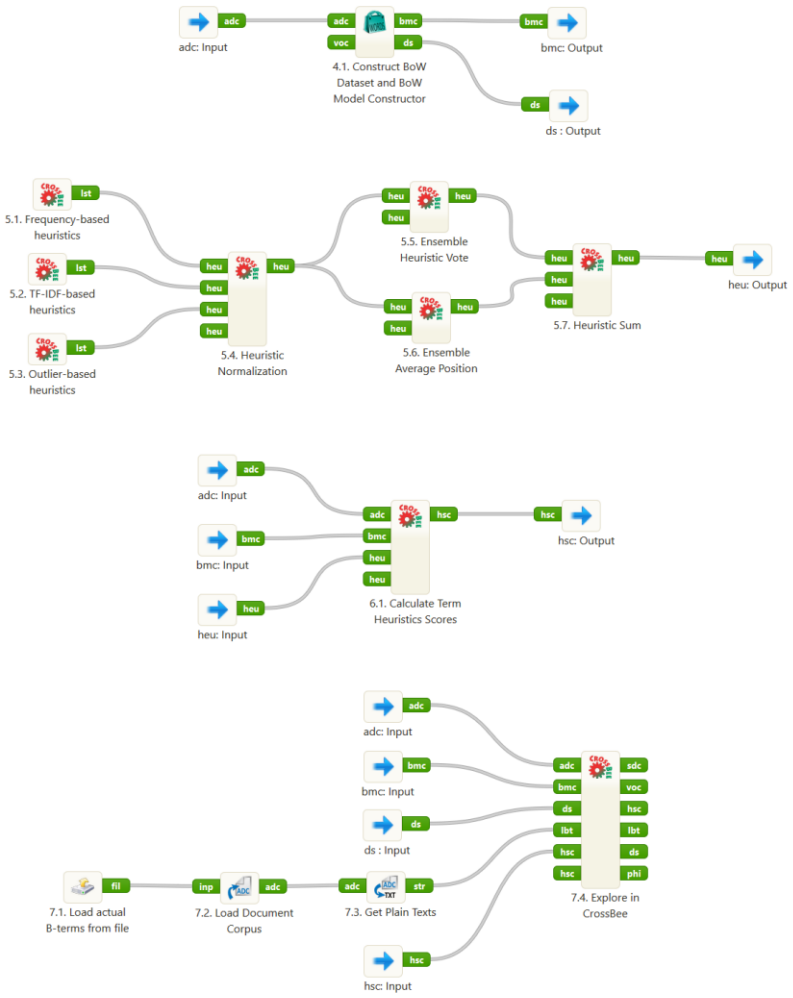

Figure 7. Workflows for the last four sub-processes that are implemented by using CrossBee components in TextFlows.

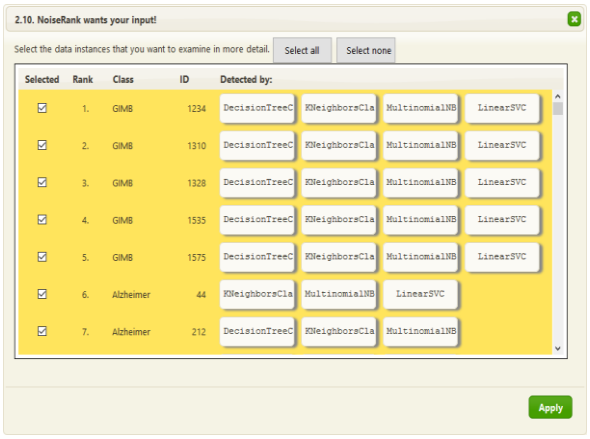

Figure 8. User interface for NoiseRank widget for inspecting and selecting outlier documents.

Conclusions and further work

This report presents the recent developments of the ClowdFlows platform by using the latest software technologies allows users to present complex procedures as a sequence of simple steps. This makes the platform usable for non-experts. The ClowdFlows platform allows importing web services as workflow components. Moreover, as a web application, the ClowdFlows platform poses no software requirements and can be used from any modern browser, including mobile devices.

Two use cases of implemented workflows were described. As a first use case we have developed a workflow for evaluating and comparing several machine learning algorithm implementations, while within second use case we have presented a workflow for finding interesting subsets of patients from the breast cancer dataset. All constructed workflows can be declared either as private or public, which enables sharing the developed solutions, data and results on the web and in scientific publications.

We have finalized the implementation of TextFlows in 2016, a fork of ClowdFlows, which is a specialized visual programming platform for text mining and natural language processing. The use case relevant for the project HinLife was also presented in this report. In future work we plan to introduce several improvements to ClowdFlows and TextFlows that will further benefit widget developers as well as end-users.

The ClowdFlows core and its widgets are currently maintained in a common code repository, which means that the widget packages cannot be developed independently of the ClowdFlows core functionality. This design was not problematic in the project's infancy, but the widget repository has now grown considerably (at the time of writing: 33 packages containing 490 widgets). This is inflexible, as all changes to the widgets must be approved by the core maintainers before they are included to the main repository. To address this, we will extract widget packages into separate Python packages in their own code repositories. This has several benefits: a) we can focus on developing the ClowdFlows core functionality, b) users can create new widget packages completely independently, as well as contribute to existing packages, c) we no longer need to maintain separate branches such as TextFlows, because we will also extract TextFlows-specific widgets into their own packages. TextFlows then becomes a specific ClowdFlows installation that includes TextFlows packages. To support this new architecture, we need to implement a mechanism to support such "external" packages, as well as effort to create separate package repositories.

The second major architectural change (but completely independent from the previous) is the new frontend implementation. We will move from the current tightly coupled frontend implementation, to a more flexible approach using modern Javascript technologies for implementing complex web-based applications (such as WebSockets and frameworks like React, Angular2 or Ember), which were not available when the ClowdFlows project began. The new frontend will be completely independent from the backend (executing, storing, editing workflows, etc.) and will communicate via a REST[5] API implemented for this purpose. This change has several implications: a) better performance since all of the execution logic will be moved to the backend, b) the frontend can be reused for any backend that implements the REST API (e.g., a new backend for big data using Spark), c) implement a different frontend for the same backend (e.g., for a mobile application or a specific use case), d) easier to maintain frontend.

We also plan to include several new features and address issues that were brought to our attention by the current users of the platform. These include: a) a system for recommending widgets to add to the workflow based on a database of existing workflows, b) optimized reads/writes of data to support handling of larger datasets and faster execution, c) many minor user interface improvements (better integrated documentation, new canvas operations like "move to subprocess", etc.).

All of the previously described changes will be released together in ClowdFlows 2.0. Like the current version, 2.0 will be available as part of the public installation[6], as well as on GitHub[7].

[5] Representational state transfer
[6] http://clowdflows.com
[7] https://github.com/xflows/

References

[1] ClowdFlows: Online workflows for distributed big data mining. J Kranjc, R Orač, V Podpečan, N Lavrač, M Robnik-Šikonja. Future Generation Computer Systems 68, 38-58, 2016.

[2] TextFlows: A visual programming platform for text mining and natural language processing. M Perovšek, J Kranjc, T Erjavec, B Cestnik, N Lavrač. Science of Computer Programming 121, 128-152, 2016.

[3] M. Lichman, UCI Machine Learning Repository, 2013. URL http://archive.ics. uci.edu/ml.

[4] B. Sluban, D. Gamberger, N. Lavrač, Ensemble-based noise detection: noise ranking and visual performance evaluation, Data Min. Knowl. Discov. 1–39, 2013.

[5] Gene Ontology: http://www.geneontology.org/

[6] KEGG: Kyoto Encyclopedia of Genes and Genomes. Kanehisa M, Goto S. Nucleic Acids Res 28, 27–30, 2000.