Future Generation Computer Systems 68 (2017) 38–58

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

ClowdFlows: Online workflows for distributed big data mining

Janez Kranjc[a,b,*], Roman Orač[c], Vid Podpečan[a], Nada Lavrač[a,b,d], Marko Robnik-Šikonja[c]

[a] *Jožef Stefan Institute, Ljubljana, Slovenia*
[b] *Jožef Stefan International Postgraduate School, Ljubljana, Slovenia*
[c] *Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia*
[d] *University of Nova Gorica, Nova Gorica, Slovenia*

HIGHLIGHTS

- We present a cloud based platform for big data and stream processing with workflows.
- The ClowdFlows platform enables processing of multiple concurrent data streams.
- Several machine learning algorithms were implemented in the map-reduce paradigm.
- Using all data in distributed mode is better than using a subset in non-distributed.
- The ClowdFlows platform handles big data sets with nearly perfect linear speedup.

ARTICLE INFO

Article history:
Received 1 February 2016
Received in revised form
14 June 2016
Accepted 4 July 2016
Available online 8 September 2016

Keywords:
Data mining platform
Cloud computing
Scientific workflows
Batch processing
Map-reduce
Big data

ABSTRACT

The paper presents a platform for distributed computing, developed using the latest software technologies and computing paradigms to enable big data mining. The platform, called ClowdFlows, is implemented as a cloud-based web application with a graphical user interface which supports the construction and execution of data mining workflows, including web services used as workflow components. As a web application, the ClowdFlows platform poses no software requirements and can be used from any modern browser, including mobile devices. The 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. The server-side software of ClowdFlows can be multiplied and distributed to any number of computing nodes. From a developer's perspective the platform is easy to extend and supports distributed development with packages. The paper focuses on big data processing in the batch and real-time processing mode. Big data analytics is provided through several algorithms, including novel ensemble techniques, implemented using the map-reduce paradigm and a special stream mining module for continuous parallel workflow execution. The batch mode and real-time processing mode are demonstrated with practical use cases. Performance analysis shows the benefit of using all available data for learning in distributed mode compared to using only subsets of data in non-distributed mode. The ability of ClowdFlows to handle big data sets and its nearly perfect linear speedup is demonstrated.

1. Introduction

Computer-assisted data analysis has come a long way since its humble beginnings on first digital computers with stored programs. In his 1962 paper entitled "The future of data analysis" [1] J. W. Tukey stated that the availability of electronic computers for some tasks is surprisingly "important but not vital" and "is vital" for others. Without doubt, the situation has changed profoundly and any serious attempt to mine knowledge from real world data must take advantage of modern computing methods and modern computer organization. However, while most scientists claim that the size of data and their rate of production is one of today's main challenges in data mining [2], it is often forgotten that developments in the field of data analysis have also produced an almost unimaginable amount of methods, algorithms, software and architectures. They are available to anyone, but their complexity and specific requirements prevent the general public and also research specialists to use them effectively without knowing the internal details. This problem has

* Corresponding author at: Jožef Stefan Institute, Ljubljana, Slovenia.
E-mail address: janez.kranjc@ijs.si (J. Kranjc).

http://dx.doi.org/10.1016/j.future.2016.07.018
0167-739X/© 2016 Elsevier B.V. All rights reserved.

been recognized years ago [3]; in his seminal work John Chambers noted that "there should not be a sharp distinction between users and programmers" and that language, objects, and interfaces are key concepts that make computing with data effective [4].

The development of modern programming languages, programming paradigms and operating systems initiated research on computer platforms for data analysis and development of modern integrated data analysis software. Such platforms offer a high level of abstraction, enabling the user to focus on the analysis of results rather than on the ways to obtaining them. In the beginning, single algorithms were implemented as complete solutions to specific data mining problems, e.g., the C4.5 algorithm [5] for the induction of decision trees. Second generation systems like SPSS Clementine, SGI Mineset, IBM Intelligent Miner, and SAS Enterprise Miner were developed by large vendors, offering solutions to typical data preprocessing, transformation and discovery tasks, and also providing graphical user interfaces [6]. Many of the later developments took advantage of the operating system independent languages such as the Java platform to produce complete solutions, which also include methods for data preprocessing and visual representation of results [7].

Visual programming [8], an approach to programming, where a procedure or a program is constructed by arranging program elements graphically instead of writing the program as a text, has become widely recognized as an important element of an intuitive interface to complex data mining procedures. All modern advanced knowledge discovery systems offer some form of workflow construction and execution, as this is of crucial importance for conducting complex scientific experiments, which need to be repeatable, and easy to verify at an abstract level and through experimental evaluation. The standard data mining platforms like Weka [7], RapidMiner [9], KNIME [10] and Orange [11] provide a large collection of generic algorithm implementations, usually coupled with an easy-to-use graphical user interface. While the operating system independence and the ability to execute data mining workflows was the most distinguished feature of standard data mining platforms a decade ago, today's data mining software is confronted with the challenge how to make use of newly developed paradigms for big data processing and how to effectively employ modern programming technologies.

This paper presents a data mining platform called ClowdFlows [12] and focuses on its capabilities of big data processing. ClowdFlows was developed with the aim to become a new generation platform for data mining, using the latest technologies in the implementation. It implements the following advanced features.

Most notably, the ClowdFlows platform runs as a web application and poses no software requirements to the users—e.g., ClowdFlows can also be accessed from modern mobile devices. Its server side software is easily multiplied and distributed to any number of processing nodes. As a modern data mining platform, ClowdFlows supports interactive data mining workflows which are composed, inspected and executed by using the ClowdFlows web interface. ClowdFlows workflows can be private or public, the later offer a unique way of sharing implemented solutions in scientific publications thus solving the Executable paper challenge [13]. Web services are supported and can be used as workflow components in an intuitive way. Processing of big data in batch mode is enabled by integrating the Disco framework, a lightweight, open-source framework for distributed computing using the map reduce paradigm [14]. In order to process big data in batch mode, we have developed a new machine learning library for the Disco MapReduce framework and included it in ClowdFlows. The library includes several standard algorithms as well as new ensemble techniques, which we evaluate and show the benefit of exploiting all available data in distributed setting compared to using only subsamples in a single node. Finally, ClowdFlows is developer-friendly as its server-side is written in Python, easily extensible, and supports distributed development using packages.

The ClowdFlows platform is released under an open source license and is publicly available on the web [12]. Users can choose either to use the publicly deployed version available at http://clowdflows.org, or clone the sources and deploy the system on their own machine or cluster of machines [15].

The rest of the paper is structured as follows. Section 2 presents the related work on data mining platforms and modern data processing paradigms. In Section 3 we present a motivational use case followed by the presentation of the design and implementation of the ClowdFlows platform. Section 4 presents the real-time analysis features of the ClowdFlows platform and demonstrates the ClowdFlows stream mining capabilities with a use case of dynamic semantic analysis of news feeds. The processing of big data in batch mode and the development of a machine learning library for the Map Reduce paradigm is presented in Section 5 which also includes a practical use case. Section 6 presents newly developed distributed random forest based ensemble methods. The batch processing mode is evaluated and validated in Section 7. In Section 8 the ClowdFlows platform is compared to related platforms and options for their integration are presented. Section 9 summarizes the work and concludes the paper by suggesting directions for further work. In the Appendix, summation form algorithms in DiscoMLL are described.

2. Related work

Visual construction and execution of scientific workflows is one of the key features of the majority of current data mining software platforms. It enables the users to construct complex data analysis scenarios without programming and allows easy comparison of different options. All early major data mining platforms, such as Weka [7], RapidMiner [9], KNIME [10] and Orange [11] support workflow construction. The most important common feature is the implementation of a *workflow canvas* where complex workflows can be constructed using drag, drop and connect operations with available components. The range of available components typically includes database connectivity, data loading from files and pre-processing, data and pattern mining algorithms, algorithm performance evaluation, and interactive and non-interactive visualizations.

Even though such data mining software solutions are user-friendly and offer a wide range of components, some of their deficiencies severely limit their utility. Firstly, all available workflow components are specific and can be used in only one platform. Secondly, the described platforms are implemented as standalone applications and have specific hardware and software dependences. Thirdly, in order to extend the range of available workflow components in any of these platforms, knowledge of a specific programming language is required. This also means that they are not capable of using existing software components, implemented as web services, freely available on the web.

In order to benefit from service-oriented architecture concepts, software tools have emerged, which are able to use web services and access large public databases. Environments such as Weka4WS [16], Orange4WS [17], Web Extension for RapidMiner and Taverna [18] allow the integration of web services as workflow components. However, with the exception of Orange4WS and Web Extension for RapidMiner, these environments are mostly focused on specific scientific domains such as systems biology, chemistry, medical imaging, ecology and geology and do not offer general purpose machine learning and data mining algorithm implementations.

Remote workflow execution (on machines different from the one used for workflow construction) is employed by KNIME Cluster execution and RapidMiner using the RapidAnalytics server. This allows the execution of local workflows on more powerful machines and data sharing with other users, with the requirement that the client software is installed on the user's machine. The client software is used for designing workflows, which are executed on remote machines, while only the results can be viewed using a web interface.

All the above mentioned platforms are based on technologies that are becoming legacy and do not benefit from modern web technologies, which enable truly independent software solutions. On the other hand, web-based workflow construction environments exist, but they are mostly too general and not coupled to any data mining library. For example, Oryx Editor [19] can be used for modeling business processes and workflows, while the genome analysis tool Galaxy [20] (implemented as a web application) is limited to workflow components provided by the project itself. An exception is the ARGO project [21], where the aim was to develop an online workbench for analyzing textual data based on a standardized architecture (UIMA), supporting interactive scientific workflow construction and user collaboration through workflow sharing, providing a selection of data readers, consumers and some components for text analytics (mostly tagging, annotation and feature extraction). Finally, the OnlineHPC web application, which is based on the Taverna server as the execution engine, offers an online workflow editor, which is mostly a user friendly interface to Taverna.

Grid workflow systems such as Pegasus [22], DAGMan [23] and ASKALON [24] were developed with the aim of simplifying intensive scientific processing of large amounts of data where the emphasis is on distribution of independent command line applications (grid jobs or tasks) and summarization of results. As the interactive analysis and graphical interfaces are not their most important features, some of them do not implement graphical interface to workflows but provide flexible programming interfaces instead. These platforms contain one or more grid middleware layers, which enable the execution on computer grids such as HTCondor and Globus.

Some platforms support more than one model of computation (see the analysis of workflow interoperability by Elmroth et al. [25]) and enable the use of web services, grid services as well as other execution environments (e.g., custom modules via foreign language interfaces). Kepler [26], Triana [27] and Taverna [18] are the most well known examples of such platforms.

As a response to the ever increasing amount of data several new distributed software platforms have emerged. In general, such platforms can be categorized into two groups: batch data processing and data stream processing.

A well known example of a distributed batch processing framework is Apache Hadoop [28], an open-source implementation of the MapReduce programming model [14] and a distributed file system called Hadoop Distributed Filesystem (HDFS). It is used in many environments and several modifications and extensions exist, also for online (stream) processing [29] (e.g., parallelization of several learning algorithms using an adaptation of MapReduce is discussed by Chu et al. [30]). Apache Hadoop is also the base framework of Apache Mahout [31], a machine learning library for large data sets, which currently supports recommendation mining, clustering, classification and frequent itemset mining. A more recent alternative to Hadoop is Apache Spark. Spark was developed to overcome Hadoop's shortcoming that it is not optimized for iterative algorithms and interactive data analysis, which performs multiple operations on the same set of data [32]. Radoop [33], a commercial big data analytics solution, is based on RapidMiner and Mahout, and uses RapidMiner's data flow interface. The Disco project [34] that we use is an alternative to Apache Hadoop. It is a lightweight open source framework for distributed computing based on the MapReduce paradigm and written in Erlang and Python.

For data stream processing, two best known platforms are S4 [35] and Storm [36]. The S4 platform is a fully distributed real-time stream processing framework. The stream operators are defined by the user code and the configuration jobs described with XML. Storm is a stream processing framework that focuses on guaranteed message processing. The user constructs workflows in different programming languages such as Python, Java, or Clojure. Neither of these two platforms features an easy to use graphical user interface.

SAMOA [37] is an example of a new generation platform that targets processing of big data streams. In contrast to distributed data mining tools for batch processing using MapReduce (e.g., Apache Mahout), SAMOA features a pluggable architecture on top of S4 and Storm for performing common tasks, such as classification and clustering. The platform does not support visual programming with workflows. MOA (Massive On-line Analysis) is a non-distributed framework for mining data streams [38]. It is related to the Weka project and a bi-directional interaction of the two is possible. MOA does not support visual programming of workflows but the ADAMS project [39] provides a workflow engine for MOA, which uses a tree-like structure instead of an interactive canvas.

Sharing data and experiments has been implemented in the Experiment Database [40], which is a database of standardized machine learning experimentation results. Instead of a workflow engine it features a visual query engine for querying the database, and an API for submitting experiments and data.

Substantial efforts have also been invested in developing systems for streamlining experimentation and data analysis in multi agent based systems, particularly for game-playing, in distributed systems [41]. In such systems distributed-computing applications can be customized by a human monitoring expert who controls the execution of an experiment through a web-based graphical user interface.

More recent additions to the family of machine learning software are Google's TensorFlow [42] and the Machine Learning Service from Microsoft Azure [43]. TensorFlow is an implementation for executing machine learning algorithms on thousands of computational devices such as GPU cards. The system can be used to express a wide variety of algorithms for problems in speech recognition, computer vision, robotics, and natural language processing. However, the system lacks a conventional graphical user interface and is invoked as a software library. The Machine Learning Service from Microsoft Azure provides an easy to use and intuitive graphical user interface to construct data mining workflows on a canvas, but it is proprietary and requires the user to subscribe to Microsoft's cloud services.

3. ClowdFlows platform

Software technologies that were used to implement the ClowdFlows platform allow an easy integration of very diverse programming libraries and algorithm implementations. Various wrappers allow the Python programming language environment to connect to software written in Java, C, C++, C#, Fortran, etc. Several libraries for web service interoperability are also available. The ClowdFlows platform currently integrates three major machine learning libraries: Weka [7], Orange [11] and scikit-learn [44]. Integration of Orange and scikit-learn is native as both are written in Python/C++. Weka algorithms are implemented as web services using the JPype [45] wrapper library.

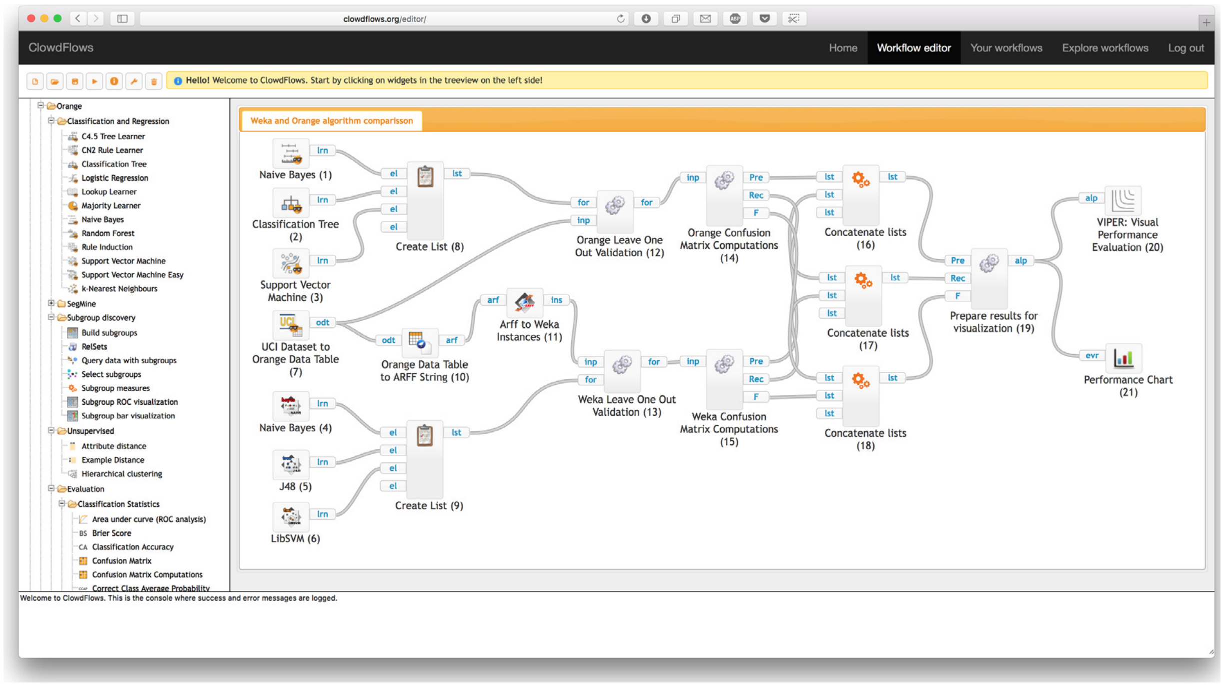

clowdflows.org/editor/
ClowdFlows
Home
Workflow editor
Your workflows
Explore workflows
Log out
Hello! Welcome to ClowdFlows. Start by clicking on widgets in the treeview on the left side!
Orange
Classification and Regression
C4.5 Tree Learner
CN2 Rule Learner
Classification Tree
Logistic Regression
Lookup Learner
Majority Learner
Naive Bayes
Random Forest
Rule Induction
Support Vector Machine
Support Vector Machine Easy
k-Nearest Neighbours
SegMine
Subgroup discovery
Build subgroups
RelSets
Query data with subgroups
Select subgroups
Subgroup measures
Subgroup ROC visualization
Subgroup bar visualization
Unsupervised
Attribute distance
Example Distance
Hierarchical clustering
Evaluation
Classification Statistics
Area under curve (ROC analysis)
Brier Score
Classification Accuracy
Confusion Matrix
Confusion Matrix Computations
Correct Class Average Probability
Weka and Orange algorithm comparisson
Naive Bayes (1)
Classification Tree (2)
Support Vector Machine (3)
UCI Dataset to Orange Data Table (7)
Naive Bayes (4)
J48 (5)
LibSVM (6)
Create List (8)
Create List (9)
Orange Data Table to ARFF String (10)
Arff to Weka Instances (11)
Orange Leave One Out Validation (12)
Weka Leave One Out Validation (13)
Orange Confusion Matrix Computations (14)
Weka Confusion Matrix Computations (15)
Concatenate lists (16)
Concatenate lists (17)
Concatenate lists (18)
Prepare results for visualization (19)
VIPER: Visual Performance Evaluation (20)
Performance Chart (21)
Welcome to ClowdFlows. This is the console where success and error messages are logged.

Fig. 1. 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/.

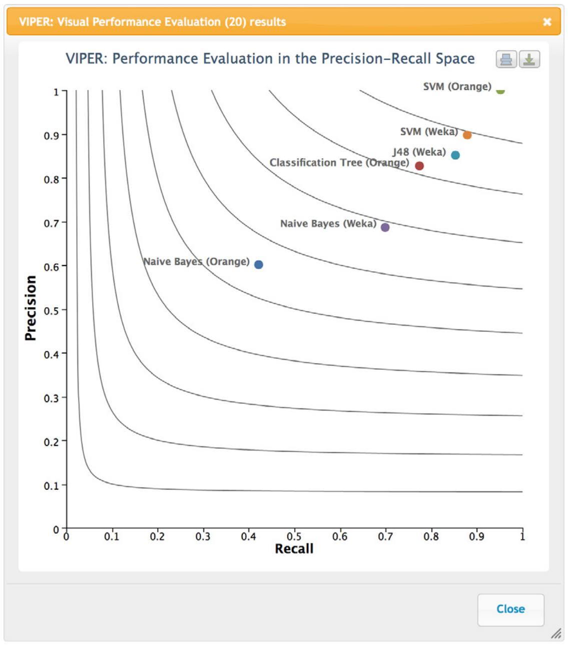


Fig. 2. Visual performance evaluation of several machine learning algorithms implemented in ClowdFlows.

3.1. Illustrative example

We first demonstrate the use of the platform with an illustrative use case followed by the design and architecture of ClowdFlows.

The goal of this simple use case is to present a few basic features of ClowdFlows. To this end, 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 [46] and the results of the evaluation are presented using the VIPER (Visual Performance Evaluation) [47] 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 Fig. 1. 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 Fig. 2.

The public ClowdFlows installation features many other example workflows including workflows that demonstrate regression[1] and clustering.[2]

3.2. Platform architecture and technologies

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 Fig. 3. The platform consists of the following components: a graphical user interface, a core processing server, a database, a

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

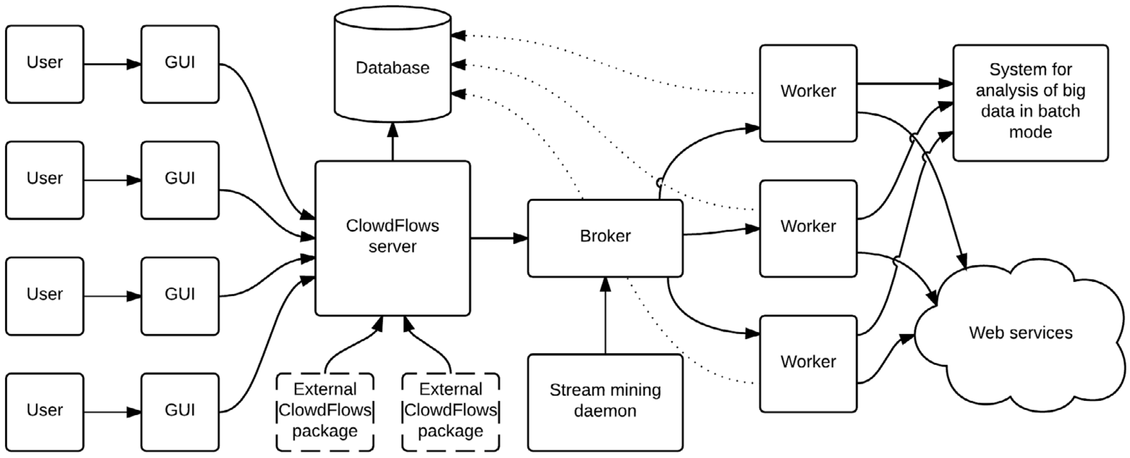


Fig. 3. An overview of the ClowdFlows platform architecture. A similar figure (without the big data analytics components) has appeared in our previous publication [48].

stream mining daemon, a broker that delegates execution tasks, distributed workers, web services, and a module for big data analysis in batch mode.

3.2.1. Graphical user interface

Users interact with the platform primarily through the graphical user interface in a web browser. We implemented the graphical user interface in HTML and JavaScript, with an extensive use of the jQuery library [49]. The jQuery library was designed to simplify client-side scripting, and is the most popular JavaScript library in use [50]. The user interface is served from the primary ClowdFlows server.

As illustrated in Fig. 1, the graphical user interface provides a workflow canvas where workflow components (widgets) can be added, deleted, re-positioned and connected with one another to form a coherent workflow. The graphical user interface is synchronized with the ClowdFlows server using asynchronous HTTP requests, which notify the server of any and all user actions.

The job of the graphical user interface is to also render results in a meaningful representation. Each widget that can produce visualized results does so by sending the represented data in HTML and JavaScript to the graphical user interface, which in turn shows it to the user in a non-obtrusive pop-up dialog.

Aside from the workflow construction and results visualization capabilities, this layer of the application also displays a list of public workflows that can be copied by the currently logged in user.

All the graphical user interface code resides on the server but is executed in the users' browsers.

3.2.2. ClowdFlows server

The ClowdFlows server software is written in Python and uses the Django web framework [51]. The Django framework is a high level Python web framework that encourages rapid development and provides an object-relational mapper and a powerful template system.

The ClowdFlows server consists of a web application and the widget repository.

The web application defines the models, views, and templates of ClowdFlows. 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 their 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 stores the results of the task to its outputs. Connections are used to transfer data between two widgets and may exist only between an output of a widget and an input to another widget. Data is transferred through connections, so inputs can only receive data from connected outputs. Parameters are similar to inputs, but need to be entered by the user. Inputs can be transformed into parameters and vice-versa, depending on the user's needs.

Data from the data model are stored in the database. It should be noted that there are two representations of widgets in the data model and database. The *abstract widget* is a description of a specific widget in the repository and holds no other information than the inputs, outputs, parameters and function of the widget. The non-abstract widget is a separate entity that represents a particular instance of an abstract widget and contains information about the data inputs and outputs, and its spatial position in a specific workflow. To summarize, when the user constructs a workflow, she chooses from a set of abstract widgets to create instances of non-abstract widgets that can process data. This design decision was made to allow user customization of widgets and to ensure the functionality of workflows, even when the abstract widgets change over time.

The ClowdFlows application also implements a workflow execution engine which executes widgets in the workflow in the correct order. The engine issues tasks to the workers to execute widgets. Initially only the widgets that have no predecessors are executed (a predecessor is a widget that is connected on the input of a widget). Once these widgets have successfully executed the engine searches for widgets whose predecessors have been successfully executed and issues tasks to the workers to execute them. When there are no more widgets to execute, a workflow is considered successfully executed.

The ClowdFlows widget repository consists of four groups of widgets: regular widgets, visualization widgets, interactive widgets and workflow control widgets.

Regular widgets are widgets that take data on the input and return data on the output. Visualization widgets do the same with the addition that they provide an HTML/JavaScript view which can be rendered by the graphical user interface to display results. Interactive widgets are widgets that serve a view to the user during execution time. Interactive widgets can show data on the input to the user and can process the user interaction in its function which affects the data on the output.

Workflow control widgets are special widgets that allow creation of subworkflows (workflows encapsulated in widgets that contain a workflow), creation of *for loops* and special types of for loops that are used for cross validation. With the workflow control widgets, the workflows can also be exposed as REST API services. These REST API services provide HTTP endpoints that can be called from anywhere to invoke the execution of a workflow and fetch the results.

The functions of regular widgets, visualization widgets, and interactive widgets are stored in the widgets libraries. The widget libraries are packages of functions that are called when a widget is executed. These functions define the functionality of each particular widget.

By default, ClowdFlows comes with a set of widgets that can be expanded. The initial set of widgets encompasses solutions to many data mining, machine learning and other tasks such as: classification, clustering, regression, association rule learning, noise detection, decision support, text analysis, natural language processing, inductive logic programming, graph mining, visual performance evaluation, and others.

3.2.3. Database

The database is the part of the system that stores all the information about the workflows and all the user uploaded data.

The object-relational mapper implemented in Django provides an API that links objects to a database, which means that the ClowdFlows platform is database agnostic. PostgreSQL, MySQL, SQLite and Oracle databases are supported. MySQL is used in the public installation of ClowdFlows. The database installation can be deployed on a cluster to ensure scalability of the system.

3.2.4. Worker nodes

Worker instances are instances of the ClowdFlows server that do not serve the graphical user interface and are only accessed by a broker that delegates execution tasks. They execute workflows and workflow components. The workers report success or error messages to the broker and feature timeouts that ensure fault tolerance if a worker goes offline during the run-time. The number of workers is arbitrary and they can be connected or disconnected during run-time to ensure scalability and robustness. Workers subscribe to the message broker system, which can be deployed on multiple machines. The ClowdFlows system offers support for several message broker systems. RabbitMQ [52] is used in the ClowdFlows public installation.

3.2.5. Web services

In order to allow consumption of web services and import them as workflow components, the PySimpleSoap library [53] is used. PySimpleSoap is a light-weight library written in Python and provides an interface for client and server web service communication, which allows importing WSDL (Web Service Definition Language) web services as workflow components, and exposing entire workflows as WSDL web services.

3.2.6. Scaling and process distribution over cloud resources

The ClowdFlows platform can scale horizontally in a very straight forward way. The four components that need to be scaled are the ClowdFlows server, the database, the broker, and the worker nodes.

Scaling the ClowdFlows server is done simply by installing it on multiple machines and running it behind a web server with load balancing capabilities such as Nginx. Horizontally scaling the ClowdFlows server is required when there are many simultaneous users accessing the platform at once. The requests are then routed round robin to different ClowdFlows server instances to reduce the load.

As the ClowdFlows platform is database agnostic it is entirely dependent on the scaling ability of the selected database software. Popular database solutions such as MySQL and PostgreSQL can be transformed into distributed scaled-out systems, however as the ClowdFlows platform does not normally perform demanding database operations (apart from simple insertions and selections) this is not likely to require scaling.

The scalability of the broker also depends on the choice of its implementation. Both RabbitMQ and Redis, which are popular broker solutions, can be easily deployed into clusters where nodes are added and removed. Scaling the broker is required if the data that passes from one workflow component to another is large.

Similarly to the ClowdFlows server, the worker nodes (which are just headless instances of the ClowdFlows server) can easily be executed in parallel on multiple machines as long as they all connect to a single broker (or broker cluster). The broker ensures that tasks are distributed evenly across the workers. Increasing the number of workers is by far the most frequently required scaling operation. Each worker can execute a set number of widgets in parallel at any given time. If there are more workflows executed than available workers some workload will be delayed until workers finish tasks of active workflows.

The scaling of the stream mining deamon and the batch processing module is handled separately and is explained in Sections 4 and 5, respectively.

3.3. Public workflows

Since workflows in the ClowdFlows platform are processed and stored on remote servers they can be accessed from anywhere over the internet. By default, each workflow can only be accessed by its author. We have implemented an option that allows users to create public versions of their workflows.

The ClowdFlows platform generates a URL for each workflow that is defined as public. Users can share their workflows by publicizing this URL. Whenever a public workflow is accessed by the user, a copy of the workflow is created on the fly and added to the user's private workflow repository. The workflow is copied with all the data to ensure the reproducibility of experiments. Each such copied public workflow can be edited, augmented or used as a template to create a new workflow, which can be made public as well.

3.4. Extensibility and widget development

There are two ways to add widgets to the ClowdFlows platform. A widget can be either implemented as a Python function and included in a ClowdFlows package or be manually imported via the graphical user interface as a WSDL Web service.

In the ClowdFlows platform the widgets are grouped into packages. Each package consists of a set of widgets with common functionalities. A package can be bundled with the code of the platform or released as a separate Python package, which can be installed on demand. An example of such a package is the Relational Data Mining (RDM) package for ClowdFlows.[3] This package exists as a stand alone Python package but can also be included in ClowdFlows. Upon doing so the widgets are automatically discovered by the ClowdFlows platform and added to the repository.

Creating a custom package for ClowdFlows requires the developer to have ClowdFlows installed locally. The local installation of ClowdFlows provides several command line utilities for dealing with packages. These utilities create bare-bones packages that include skeleton code for widgets. A ClowdFlows widget is a Python function that receives a dictionary of widget inputs on the input and returns a dictionary of outputs as its output. The body of the function needs to be filled in by the developer of the widget to transform the inputs into the outputs as expected. The widget's inputs and outputs need to be described in the JSON format in order to be shared with other installations of ClowdFlows (including

[3] https://github.com/xflows/rdm.

the public installation). This JSON file can be written manually or by using the ClowdFlows administration interface, which provides simple forms where widget details can be entered. The JSON files are then generated from the database using command line utilities bundled with the local installation of ClowdFlows.

In a multiple worker setting each worker node needs to have all the available external packages installed. Fig. 4 shows a worker node with external packages installed and its JSON description of widgets imported using the administration tools.

Another way of adding widgets is by using the graphical user interface and entering the URL of a WSDL described Web service. This can be done without altering the code of the platform which makes it easy to use. The Web service description file is consumed by ClowdFlows and parsed to determine the inputs and outputs of the functions of the Web service. Each function of a Web service is represented as a single widget in ClowdFlows that can be used and reused in any number of workflows by the user that imported the service.

3.5. Data exchange with external systems

The ClowdFlows platform provides several ways of exchanging data with external systems. We distinguish between two types of functionalities: the inward and outward interoperability.

Inward interoperability is achieved either by consuming web services and presenting them to the users as widgets of the ClowdFlows platform, or by creating widgets that call code from other systems. By default the ClowdFlows platform consumes web services that provide functionalities of the Weka platform, and provides widgets that access code and data from the Orange and scikit-learn packages.

Outward interoperability allows any workflow to be exposed as a REST API endpoint. In order to benefit from this feature each workflow can have several API Input and API Output widgets on the canvas. These widgets are linked with the inputs that the REST API endpoint receives and the JSON output of results that it should return. In this way it is possible to construct a workflow in the ClowdFlows platform and execute it without using the graphical user interface, which makes it suitable for use in external applications.

4. Real-time data stream mining

Processing of real-time data streams is enabled in ClowdFlows: a specialized stream mining deamon was implemented that continuously executes workflows in parallel with a modified workflow execution engine that implements a halting mechanism. The stream mining capabilities of the ClowdFlows platform are described below.

4.1. Stream mining workflows and stream mining deamon

Stream mining workflows are workflows that are connected to a potentially infinite source of incoming data and need to be executed whenever there is new data on the input. Due to the nature of online data sources, it is often necessary to poll a data source for new data instead of having the data source push the data to external services such as ClowdFlows. To control execution of stream mining workflows we implemented a special deamon that executes workflows at a fixed time interval and provided a functionality to halt the execution of a workflow to stream mining widgets.

The stream mining deamon is a process that runs alongside the ClowdFlows server, loops through deployed stream mining workflows and executes them. The execution is similar to the regular workflow execution with the difference that widgets may

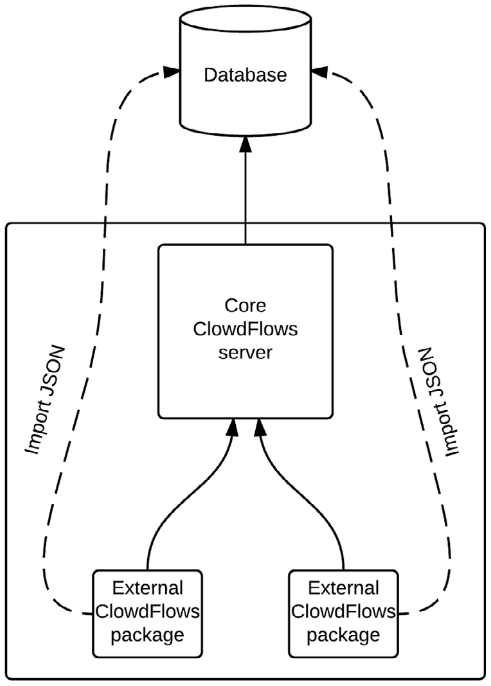


Fig. 4. A worker node of the ClowdFlows architecture with two external packages installed.

halt the execution of workflows. In practice, the workflow is executed as frequently as the data appears on the data source and produces outputs with a fixed latency depending on the workflow complexity. Stream mining workflows are, in contrast to regular workflows, executed a potentially infinite number of times until the execution is stopped by the user.

4.2. Stream mining widgets

In contrast to widgets in regular workflows, widgets in stream mining workflows have the internal memory and the ability to halt the execution of the current workflow. The internal memory is used to store information about the data stream, such as the timestamp of the last processed data instance, or an instance of the data itself. These two mechanisms were used to develop several specialized stream mining widgets.

In order to process data streams, *streaming data inputs* were implemented. Each type of stream requires its own widget to consume the stream. In principle, a streaming input widget connects to an external data stream source, collects instances of the data that it has not yet seen, and uses its internal memory to remember the current data instances. This can be done by saving small hashes of the data to preserve space or only the timestamp of the latest instance if timestamps are available in the stream. If the input widget encounters no new data instances at the stream source it halts the execution. No other widgets that are directly connected to it via its outputs will be executed until the workflow is executed again.

Several popular stream mining approaches [54] were implemented as workflow components. The *aggregation* widget was implemented to collect a fixed number of data instances before passing the data to the next widget. The internal memory of the widget is used to save the data instances until the threshold is reached. While the number of instances is below the threshold, the widget halts the execution. The internal memory is emptied and the data instances are passed to the next widget once the threshold is reached.

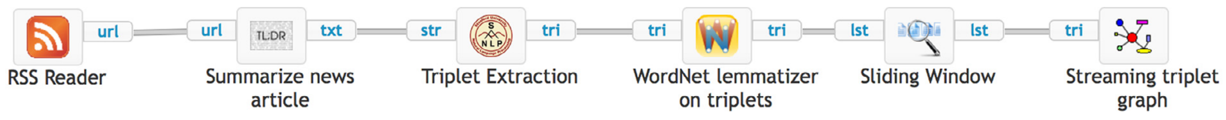


Fig. 5. The semantic triplet graph from an RSS feed workflow constructed in the ClowdFlows platform. The workflow is publicly available at http://clowdflows.org/workflow/1729/.

The *sliding window* widget is similar to the aggregation widget, except that it does not empty its entire internal memory upon reaching the threshold. Only the oldest few instances are *forgotten* and the instances inside the sliding window are released to other widgets in the workflow for processing. By using the sliding window, each data instance can be processed more than once.

Sampling widgets either pass the instance to the next widget or halt the execution, based on a condition. This condition can be dependent on the data or not (e.g., drop every second instance). The internal memory can store counters, which are used to decide which data is part of the sample.

Special *stream visualization* widgets were developed for the purpose of examining results of real-time analyses. Each instance of a stream visualization widget creates a web page with a unique URL that displays the results in various formats. This is useful because the results can be shared without having to share the actual workflows.

4.3. Illustrative use case

The aim of this use case is to construct a semantic graph from a stream of news articles in real time.

A semantic triplet graph [55] is a graph constructed from *subject–verb–object* triplets extracted from sentences. Our goal is to develop a reusable workflow that allows extraction of triplets from an arbitrary source of news articles, displays them in a two-dimensional force directed graph with words as nodes and updates them in real-time. By doing this, we transform an incoming stream of news articles into a live semantic graph that is continuously updated.

The use case is presented as a step-by-step report on how the workflow was constructed. Following this description the user can construct a fully functional workflow for an arbitrary stream of data and examine the results. In this particular workflow the Middle East section of the CNN news website is used as the incoming stream on which the semantic graph is constructed.

4.3.1. Identification and development of necessary workflow components

In order to produce a workflow that transforms an incoming stream of news articles into a semantic triplet graph we require three components: a component that connects to the RSS feed and fetches new articles as they appear, a component that extracts triplets from the articles, and a component that the triplets in the graph form. In order to create a visually appealing and useful semantic graph we decided to create three supporting widgets: a widget that fetches the article text and summarizes it by selecting five most important sentences from the article, a widget that normalizes the triplets by performing lemmatization on the extracted words, and a sliding window to force the graph to "forget" older news.

We implemented a widget that connects to an arbitrary RSS feed. This widget accepts a single parameter: the URL of the RSS feed. The internal memory of widgets was utilized to store hash codes of article URLs that have already been processed. With this we ensure that each article is only processed once. If all URLs in the feed have already been processed the widget halts the execution. The widget's single output is the URL of the article that should be processed. This widget was implemented as a Python function as explained in Section 3.4. The function has access to the argument (the URL of the RSS feed) and to the internal memory of the widget which is persistent for a particular execution of a stream. The function uses a high level Python library *requests* for fetching the data and parsing the feed.

We implemented a widget that fetches the article text from the URL, extracts the article's title and body, and summarizes it by selecting five most important sentences in the article. We developed a widget that wraps the PyTeaser library for text summarization [56]. The most important sentences are selected based on their relevance to the title and keywords, as well as the position and length of the sentences. The widget outputs the summary as a string of characters.

The triplet extraction widget implements the algorithm proposed in [57] using the Stanford Parser [58]. The widget first tokenizes the sentences and generates a parse tree for each sentence. The algorithm searches the parse tree for the subject, predicate, and object triplet. If the triplet is found, it is appended to the list of triplets that the widget returns as its output. The triplet extraction widget was implemented as a WSDL Web service which was imported into the ClowdFlows.

Normalization of words helps in building a more cohesive graph by joining similar nodes into a single node. We employ the WordNet Lemmatizer implemented in the Python NLTK library [59,60]. The lemmatization uses the WordNet's built-in morphy function. The input words are returned unchanged if they cannot be found in WordNet. This technique helps to eliminate repetitions of similar or same entities in the graph. As NLTK is a Python library, we implemented this widget as a Python function as explained in Section 3.4.

To dynamically visualize the semantic triplet graph we use a sliding window widget to keep only the 100 recent triplets. By doing this the graph only shows current news and "forgets" older news.

The visualization widget utilizes the D^3 Data-Driven Documents JavaScript library [61] to display the semantic triplet graph. The graph is constructed by creating a node for each unique word in the current sliding window. Edges are constructed from the subject–verb and verb–object connections. The graph is rendered using a force-directed algorithm [62]. The visualization is updated in real-time, new nodes and edges are created in real-time and old ones are removed from the graph. Visualization widgets also render an HTML view, which allowed us to use a JavaScript library to implement the visualization. Anything that can be displayed on a web page can be displayed as a result of a visualization widget.

All the developed widgets were added to the repository and are part of the stream mining ClowdFlows package. For the sake of simplicity the Triplet Extraction Web service call was wrapped into a Python function otherwise new installations of ClowdFlows would require users to manually import it as a Web service.

4.3.2. Constructing the workflow

We constructed the workflow using the ClowdFlows graphical user interface. Widgets were selected from the widget repository, added to the canvas and connected as shown in Fig. 5.

In the RSS reader widget we entered the URL of the CNN news section feed—http://rss.cnn.com/rss/edition_meast.rss. We

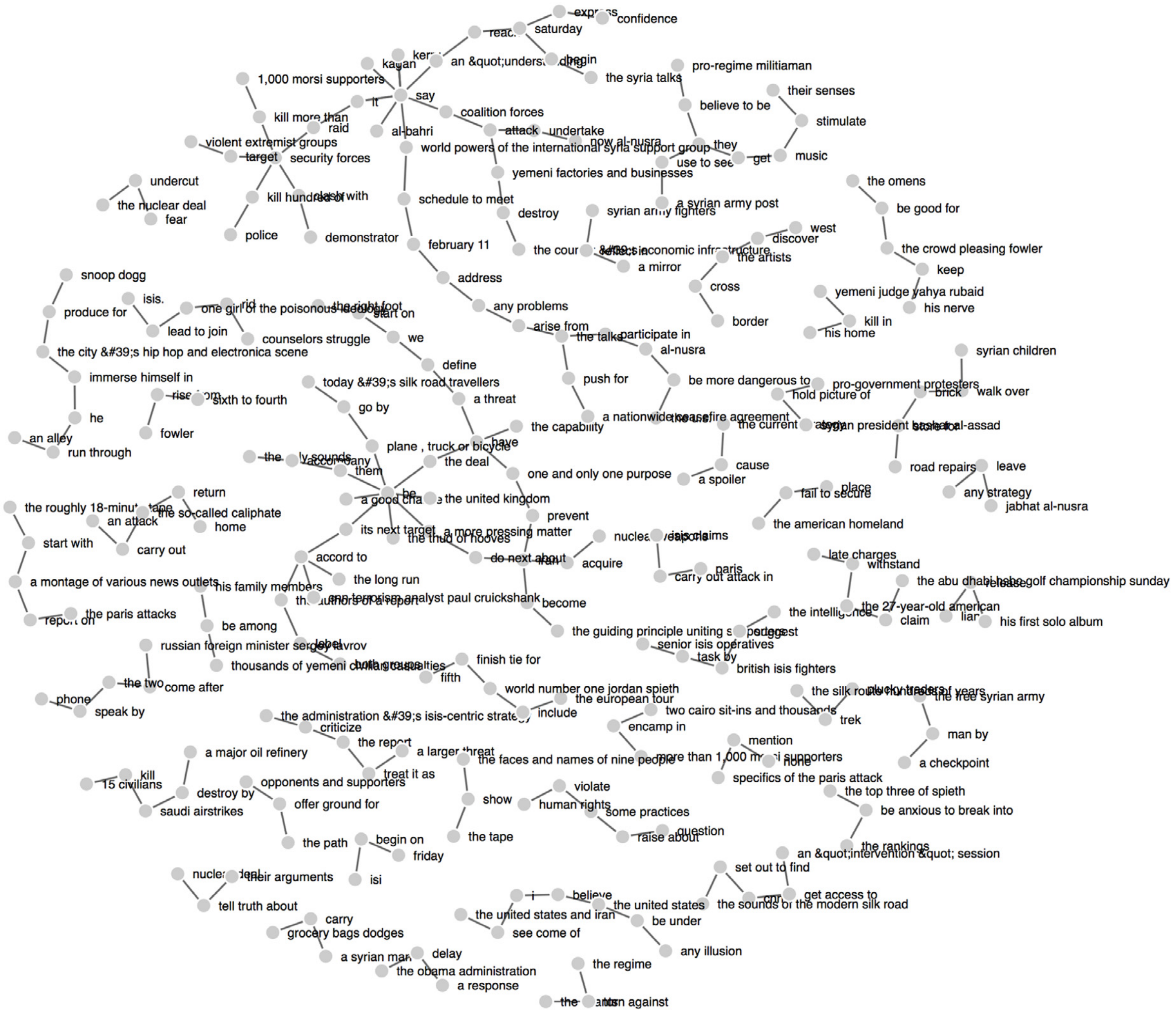


Fig. 6. The results of monitoring the Middle East edition of the CNN RSS feed. This visualization is publicly available at http://clowdflows.org/streams/data/110/63907/.

have set the size of the sliding window to 100, thereby showing the latest 100 news triplets in the graph.

We marked the workflow as public so that it can be viewed and copied. The URL of the workflow is http://clowdflows.org/workflow/1729/. By clicking the button "Start stream mining" on the workflows view (http://clowdflows.org/your-workflows/) we instructed the platform to start executing the workflow with the stream mining daemon. A web page is created with detailed information about the stream mining process. This page contains the link to the visualization page with the generated semantic graph.

4.3.3. *Monitoring the results*

By using a stream visualization widget in the workflow we can observe the results of the execution in real time. The ClowdFlows platform generates a web page for each stream visualization widget in the workflow.

Our workflow has only one stream visualization widget (see Fig. 5), therefore the ClowdFlows platform generates one web page with the results. The visualization of the CNN data stream can be found at http://clowdflows.org/streams/data/110/63907/. This visualization shows how the words in the most important sentences of multiple articles are linked to each other via the extracted *subject–verb–object* triplets. A screenshot of the semantic triplet graph is shown in Fig. 6.

The workflow presented is general and reusable. The RSS feed chosen for processing is arbitrary and can be trivially changed. The results of the workflow can be further exploited by performing additional data analysis on the graph, such as constructing a summary of several news articles or discovering links between them. Such a graph could be used to recommend related news to the reader.

4.3.4. *Evaluating the limitations of stream mining in ClowdFlows*

Stream mining workflows are executed by the Stream mining deamon, which is a separate process that exists with the sole purpose of executing stream mining workflows on a set time interval. For streams with a high rate of incoming data this time interval has to be set in such way that the rate of processing the data is higher or equal to the production rate of the data on the inputs, while producing the results with a fixed latency (the amount of time it takes to execute a workflow).

Streaming workflows usually run for a fixed amount of time. Having the execution interval (frequency) shorter than the workflow execution time (latency) means that the workflow for the same stream will be executed many times in parallel. Since ClowdFlows is a collaborative platform with many concurrent executions of stream mining workflows it is important to know the limitations of such executions so that it is possible to determine when to add new resources to ensure all data is processed. We have identified two possible scenarios when executing stream mining workflows: workflows that process data faster than the data is produced, and workflows that process data slower than it is produced.

Table 1
Average data analysis time for streaming data based on different incoming rates of data instances and different setups of worker nodes. The cells display the average processing time for each data instance in seconds (plus the standard deviation) and the relative amount of data instances processed during a minute of stream execution time.

Data spawn rate/Worker nodes	1 × 8	2 × 8	3 × 8
0.33 instances per second	3.597 ± 0.005 (100%)	3.603 ± 0.007 (100%)	3.602 ± 0.008 (100%)
1 instance per second	3.596 ± 0.006 (100%)	3.597 ± 0.009 (100%)	3.600 ± 0.014 (100%)
3 instances per second	7.702 ± 2.096 (85%)	3.599 ± 0.010 (100%)	3.600 ± 0.009 (100%)
6 instances per second	20.049 ± 8.220 (44%)	7.673 ± 2.061 (76%)	3.618 ± 0.010 (100%)

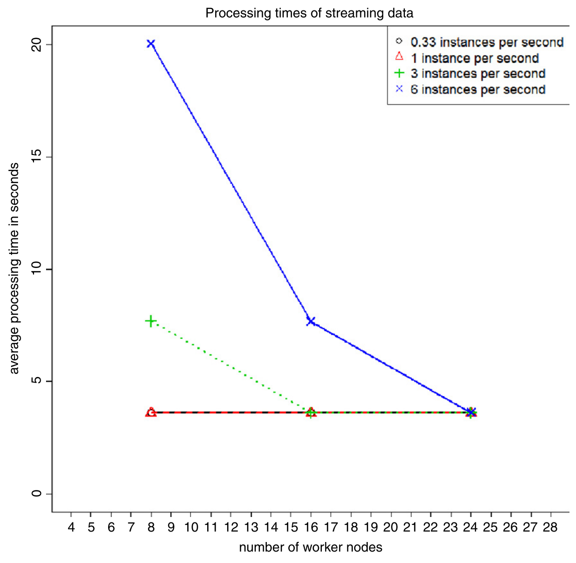


Fig. 7. Average data analysis times for streaming data based on different incoming rates of data instances over different setups of worker nodes.

For cases where the latency is shorter than the amount of time it takes for new data to appear it is clear that these workflows will never be executed in parallel. Each workflow will process the data before there is new data on the input. Even with a very small time interval the subsequent parallel executions would be immediately halted due to no data being present on the input, and thus this would not affect the resources.

We conducted an experiment where we created artificial streams with several production rates and processed them in a workflow with a latency of 3 s per data instance. We tested the stream mining capabilities of ClowdFlows against different rates of data production on the inputs: 0.33 instance per second, 1 instance per second and 3 instances per second for different setups of worker nodes. For each setting we adjusted the frequency to ensure processing of data.

We tested the platform with one, two, and three worker nodes. Worker nodes were installed on equivalent computers with 8 CPU cores. The workers were setup to work on 8 concurrent threads. We measured the processing time for processing each instance of data and calculated the relative amount of data processed in a minute in percentages. A hundred percent means that all the data on the input stream was successfully processed. The results are presented in Table 1 and displayed on the chart in Fig. 7.

The results show that using a single worker it is possible to process data at a rate three times slower than the rate of production on the input and preserve the same throughput as on the input stream. Using two workers it is possible to process data ten times slower, while a configuration with three workers allows for a twenty times slower processing rate and still cope with the demand. This allows users to have complex workflows perform analyses on the data and still see results in real time while being confident that all data was processed.

It is important to note that inter-node communication does not increase with the addition of new worker nodes. Worker nodes communicate exclusively with the broker. Tasks are issued to the broker by the stream mining daemon and results are returned to the broker by the worker nodes. Even if there is a single worker node processing the stream, the communication is always going to and from the broker. The communication delay is therefore constant.

The test also shows that the leniency for slow processing can be improved by adjusting the number of worker nodes. The worker nodes can be added and removed during runtime, which means that processing high volume streams can be resolved simply by adding more computing power to the ClowdFlows worker cluster.

5. Batch data processing with DiscoMLL library

We present the ClowdFlows system for analysis of big data in batch mode. We have chosen the Disco MapReduce framework to perform MapReduce tasks as Disco is written in Python and allows for a tighter and easier integration with the ClowdFlows platform. The downside of this choice is an apparent lack of a specialized machine learning library or toolkit within the framework, which motivated us to develop our own library with a limited but useful set of machine learning algorithms.

In this section we first introduce the MapReduce paradigm, followed by a description of the Disco Framework. We describe the implementation details of the Disco Machine Learning Library. Finally we present the integration details of batch big data processing in ClowdFlows and conclude with an illustrative use case.

5.1. MapReduce paradigm

MapReduce is a programming model for processing large data sets, which are typically stored in a distributed filesystem. Algorithms based on the MapReduce paradigm are automatically parallelized and distributed across the cluster. The user of MapReduce paradigm defines a map function and a reduce function. The map function takes an input key/value pair and generates a set of intermediate key/value pairs. Intermediate keys are grouped and passed to the reduce function. An iterator is used to access the intermediate keys and values. The reduce function merges these values and usually forms a smaller set of values. The output of a MapReduce job can be used as the input for the next job or as the result.

5.2. Disco framework

Disco is designed for storage and large scale processing of data sets on clusters of commodity server machines. It provides fault-tolerant scheduling, execution layer, and a distributed replicated storage layer. Core aspects of cluster monitoring, job management, task scheduling and distributed file system are implemented in Erlang, while the standard Disco library is implemented in Python. Activities of Disco cluster are coordinated by a central master host, which handles computational resource monitoring and allocation, job and task scheduling, log handling, and client interaction. A distributed MapReduce job executes multiple tasks, where each task runs on a single host. The job scheduler assigns resources to jobs, minimizes the data transfer over network, takes care of load balancing and handles changes in cluster topology.

Disco Distributed Filesystem (DDFS) provides a distributed storage layer for Disco. It is a tag based filesystem designed for storage and processing of massive amounts of immutable data. DDFS provides data distribution, replication, persistence, addressing and access.

5.3. *Disco Machine Learning Library*

To the best of our knowledge there is currently no Python package with machine learning algorithms based on the MapReduce paradigm for Disco, which motivated the development of the Disco Machine Learning Library (DiscoMLL) [63]. DiscoMLL is an open source library, build on NumPy [64] and the Disco framework. DiscoMLL is a part of the ClowdFlows platform and is responsible for the analysis of big batch data. This enables ClowdFlows users to process big batch data using visual programming. DiscoMLL provides many options for data set processing: multiple input data sources, feature selection, handling missing data, etc. It supports several data formats: plain text data, chunked data on DDFS and gzipped data formats. Data can be accessed locally or via file servers.

To take advantages of the MapReduce paradigm, it is necessary that algorithms have certain properties. As shown in [65], machine learning algorithms that fit the statistical query model [66] can be expressed in so called *summation form* and distributed on a multicore system. An example is an algorithm that requires statistics that sum over the data. The summation can be done independently on each core by dividing the data, assigning the computation to multiple cores and at the end aggregating the results.

All implemented algorithms have a fit phase and predict phase. The fit phase consists of map and combine tasks, which are parallelized across the cluster. Usually algorithms have one reduce task that aggregates outputs of map tasks and returns URL of the fit model, which is stored on DDFS. The learned model differs for each algorithm, since it contains the actual model and the parameters needed for the predict phase. The predict phase consists only of map tasks which are also parallelized across the cluster. The first step of the predict phase is to read the model from DDFS and pass it as parameter to map tasks. Then the data is read and processed with the model. The predict phase stores predictions on DDFS and outputs the URL.

We have implemented the following algorithms which are in summation form: Naive Bayes, Logistic Regression, K-means clustering, Linear regression, Locally weighted linear regression, and Support Vector Machines. Implementations of these algorithms can be found in the Appendix.

To assure enough algorithms with state-of-the-art performance [67] we also implemented several existing and new variants of ensemble methods adapted to distributed computing paradigm. The proposed ensembles are based on decision trees, which are not in summation form. These methods are presented in Section 6.

5.4. *Integration in the ClowdFlows platform*

The batch mode big data analysis in ClowdFlows is a separate module that is accessed by the worker nodes via an HTTP interface of the Disco cluster. This configuration is presented in Fig. 8.

To configure a Disco cluster, it is only necessary to set slave server hostnames or IP addresses, and the number of their CPU cores. All nodes of the cluster are connected to the Internet to access the input data on file servers. An input data is transferred to workers using HTTP GET requests and passed directly to map tasks. The ClowdFlows platform submits MapReduce jobs using HTTP interface via a proxy server to the Disco master host. In contrast to input data, results of MapReduce jobs are stored on DDFS and their locations are passed back to the ClowdFlows platform.

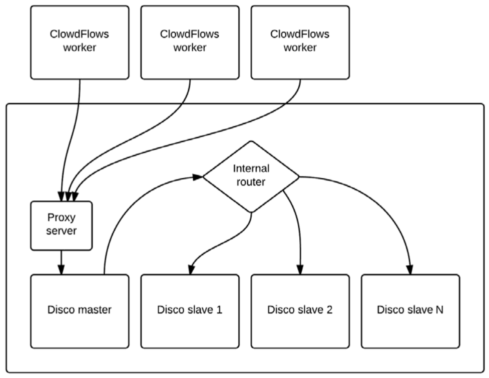


Fig. 8. An overview of the ClowdFlows system for batch mode big data analysis.

Widgets that submit MapReduce jobs were developed which allow construction of workflows for big data. Each widget calls the Disco master using an HTTP interface. The workflows designed with these widget do not differ in presentation from workflows that deal with regular data. The MapReduce paradigm and implementation methods are not obvious from the workflows themselves.

5.5. *Use case: Naive Bayes classifier for big data*

In order to demonstrate the batch big data processing mode of ClowdFlows we have implemented a simple workflow that is capable of processing data sets that do not fit into memory of conventional machines.

The aim of the use case is to build a classifier from a large test set and to use it to classify data. We first describe the implementation details of the Naive Bayes classifier in the Disco Machine Learning Library, then we follow it with a description of a ClowdFlows workflow that utilizes the DiscoMLL.

5.5.1. *Naive Bayes in Disco Machine Learning Library*

The basic form of the Naive Bayes (NB) classifier uses discrete features. It estimates conditional probabilities $P(x_j = k|y = c)$ and prior probabilities $P(y)$ from the training data, where k denotes the value of discrete feature x_j and c denotes a training label. The map function (Algorithm 1) takes the training vector, breaks it into individual features and generates output key/value pairs. Each output pair contains the training label i.e., value of y, feature index j and feature value of x_j as the key and 1 as the value. This output pair marks the occurrence of a feature value given training label. The map function also outputs the training label as the key and 1 as the value, to mark the occurrence of a training label. The map function is invoked for every training instance. The reduce function (Algorithm 2) takes the iterator over key/value pairs. Values with the same key are grouped together in the intermediate phase. If the key consist of one element, it represents an instance's label and values are summed and stored for further calculation of the prior probability. The values of other pairs are summed and output. These pairs represent the occurrences of $x_j = k \wedge y = c$ and enable calculation of conditional probabilities $P(x|y)$ in the predict phase. After all pairs are processed, prior probabilities are calculated and output. The output of the reduce function presents a model that is used in the predict phase. The outputs of the predict phase were compared with the Orange implementation of the Naive Bayes Classifier [11] and return identical results.

As an example, consider the NB classifier with the input data set in Table 2, where the target label is *Sex*. At the beginning of the

Table 2
Example data set of the human physical appearance.

Sex	Hair length	Height
M	Short	Tall
M	Short	Tall
F	Long	Medium

MapReduce job, each training instance is read and passed to the map function as its input argument (*sample*). For the first training instance, the *sample* assigns $x = [Short, Tall]$ and $y = M$. The for loop iterates through x and outputs pairs $((M, 0, Short), 1)$ and $((M, 1, Tall), 1)$. The value 1 is added to mark one occurrence of the specific feature value and training label. The map function also outputs the pair $(M, 1)$ to mark the occurrence of label M. The procedure is repeated for all training instances. Note that the first two training instances in Table 2 are the same and produce the same output pairs. Prior to invocation of the reduce function, the output pairs are grouped by the key. We get the following pairs: $((M, 0, Short), [1, 1]), ((F, 0, Long), [1]), ((M, 1, Tall), [1, 1]), ((F, 1, Medium), [1]), (M, [1, 1]), (F, [1])$. Notice that values from the first and second training instance are grouped by the key and their counts are merged in a list [1, 1]. The *iterator* over pairs is passed to the reduce function. For each key, the values are summed. The keys that mark training label occurrences are used to calculate prior probabilities, others are output in the form of $((M, 0, Short), 2)$. These values constitute the model that is used in the predict phase.

Algorithm 1: The map function of the fit phase in the NB

```
function map(sample, params)
    x, y = sample
    for j = 0 to length(x)
        #key: label, attr index and value
        #value: 1 occurrence
        output((y, j, x_j), 1)
    #mark label occurrence
    output(y, 1)
```

Algorithm 2: The reduce function of the fit phase in the NB.

```
function reduce(iterator, params)
    y_dist = hashmap()
    for key, values in iterator
        if key has 1 element
            #label frequencies
            y_dist.put(key, sum(values))
        else if key has 3 elements
            #occurrences x_j = k and y = c
            output(key, sum(values))
    prior = calculate_prior_probs(y_dist)
    output("prior", prior) #P(y)
```

For numeric attributes, a NB classifier uses a different approach to probability estimation. We provide a description of this method in the Appendix.

5.5.2. Constructing the workflow

We constructed a workflow that analyzes a big data set using the Naive Bayes machine learning algorithm based on the MapReduce paradigm.

For the purpose of this use case, we generated 6 GB of semi-artificial data [68] from the UCI image segmentation data set. The

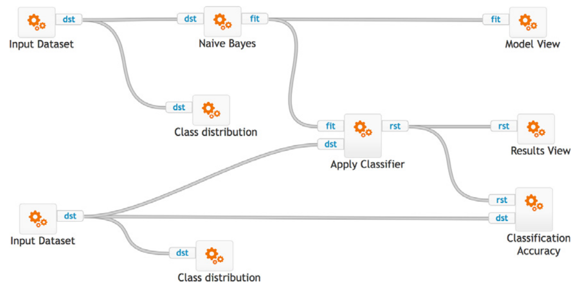


Fig. 9. The workflow for learning the NB classifier, predicting unseen instances and displaying the evaluation results. The workflow can be accessed at http://clowdflows.org/workflow/2788/.

training and testing data set each contain 3 GB of data. Both data sets were divided into chunks and loaded on a file server. The components are connected as shown in Fig. 9.

The ClowdFlows workflow consists of training a model on the image segmentation training data set with NB, using the model to predict testing instances and visualization of the results.

First, we set the input parameters specifying the image segmentation train data object. We entered multiple URLs to process the data in parallel. The feature index parameter was set to 2–21, to include all the features available in the data set. The identifier attribute was set to 0 and the target label index was set to 1 as it represents the target label. In the widget that represents the prediction data object, we entered the URLs of the corresponding chunks. The Naive Bayes widget learns the model. The Apply Classifier widget takes the prediction data and the model's location to predict the data. The Results View shows the results, the Class Distribution widget shows the distribution of labels in the data set, the Model View widget enables us to review the statistics of the model and the Classification Accuracy widget is used to calculate the accuracy of the classifier. By pressing the button "start", the workflow executes a series of MapReduce jobs on the cluster. After the workflow is finished the results are provided as a hosted file on the distributed file system.

6. Distributed ensemble methods for batch processing

Ensemble methods are known for their robust, state-of-the-art predictive performance [67]. As we want to assure high usability of ClowdFlows we developed several tree-based ensembles adapted for distributed computation with MapReduce and implemented them in DiscoMLL. We first describe the ideas behind the most successful ensemble method, random forest [69], then we review existing distributed ensembles, followed by our methods.

Random forest is one of the most robust and successful data mining algorithms [70]. It is an ensemble of randomized decision trees used as the basic classifiers. Two randomization mechanisms are used: a bootstrap sampling with replacements on the training set, separately for each tree, and random selection of a subset of attributes in each interior node of the tree. A notable consequence of using bootstrap sampling with replacement for selection of training sets is that on average $1/e \approx 37\%$ of training instances are not selected in each tree (so called out-of-bag set or OOB). This set can be used for unbiased evaluation of the model's performance and its visualization.

Tree-based ensembles can exploit distributed computing in two ways: either computing basic models independently on local subsets of instances stored in worker nodes or computing individual trees with several nodes. The first approach is used in the MReC4.5 [71] and COMET [72] systems, while the second is used in the PLANET [73]. These systems are not publicly available, so direct comparison with them is not possible.

The MReC4.5 system [71] implements a variant of bagging where the master node bootstrap samples the training instances and distributes them to local nodes, where C4.5 like decision trees [74] are constructed in the map step and returned to the master node in the reduce step. In the prediction phase all trees return their votes. The weakness of this approach is memory consumption as worker nodes operate on the data set the size of the whole data set and keep it in its internal memory.

The COMET system [72] uses non-overlapping data samples in worker nodes. During the map phase many trees are created in each node using small training sets obtained with importance-sampled voting [75]. Importance-sampled voting uses OOB set to steer the training set sampling for consecutive trees in the same worker node. A large collection of trees are returned to the master node (the reduce phase). During prediction only a subset of trees is used, depending on the agreement of already returned votes.

The PLANET system [73] constructs each tree from all data in a distributed fashion. To keep the number of map steps low and to evaluate attributes in several nodes at once it creates trees level by level instead of in a depth-first manner as usual.

We developed three tree-based ensemble methods using the MapReduce approach. To allow processing of big data we split data sets to chunks that fit into local memory of worker nodes. All three methods construct decision trees on local nodes in the map phase and gather collected trees into a forest in the reduce phase. We implemented a binary decision tree learning algorithm, which runs on a single worker and expands decision tree nodes using a priority queue. The algorithm allows different types of attribute sampling in interior tree nodes and offers several types of attribute evaluation functions, e.g. information gain [74] and MDL [76]. As the trees are constructed locally in workers we need a single map step.

In the reminder of the section we present the three methods. Their evaluation is included in Section 7.2.

6.1. *Forest of Distributed Decision Trees*

The first variant, called Forest of Distributed Decision Trees (FDDT), performs a distributed variant of bagging [77]. Instead of bootstrap sampling of the whole data set it builds decision trees on chunks of training data. In each interior decision tree node all attributes are evaluated and the best one is selected as the splitting criterion. In the prediction phase all trees are used and their majority vote is returned as a prediction.

6.2. *Distributed Random Forest*

The second variant, called Distributed Random Forest (DRF), is a distributed variant of the random forest algorithm [69]. It uses bootstrap sampling with replacement on local data chunks to construct the training sets. In each interior node a random subsample of attributes is evaluated and the best one is selected as the splitting criterion. In the prediction phase a subset of trees is randomly selected and used for prediction. If the difference between the most probable prediction and the second most probable prediction is larger than a pre-specified parameter the most probable prediction is returned, otherwise more trees are selected and the process is repeated. The process ends when the difference is large enough or all the trees have been used for prediction. This process speeds up the prediction phase by using only a small subset of trees for prediction of less difficult instances.

6.3. *Distributed Weighted Forest*

The third variant, called Distributed Weighted Forest (DWF), is based on the idea that not all trees perform equally well for each instance, so it weights the trees for each prediction instance separately, extending the idea of [78] to a distributed environment. The construction phase of randomized trees is the same as in DRF, but after each tree is constructed, it is used to predict class values of its OOB instances. If two instances are classified into the same leaf node, their similarity score is increased. In this way we get a similarity score for all instances stored in a worker node, which we can divide by the number of trees in a worker t to get a $[0, 1]$ normalized distance [69]. The distances are passed to the k-medoid clustering algorithm, which returns medoids (instances, whose average distance to all instances in the same cluster are minimal). The default value for the number of clusters k is set to $\lceil \sqrt{a} + 1 \rceil$, where a is the number of attributes. Larger values of k improve the prediction accuracy, but also increase the prediction time. Medoids are used to determine prediction reliability of trees. A reliability score used is the prediction margin, defined as a difference between the predicted probability of the correct class and the most probable incorrect class. This margin is used to weight trees during the prediction phase. The DWF algorithm returns a local (small) forest for each worker node, the medoids, and reliability scores for each tree in the local forest.

During the prediction phase we first compute the similarity between a test instance and all the medoids in all the forests using the Gower coefficient [79], which is defined for both nominal and numeric attributes. The medoids with the highest similarity to the test case are used as tree quality probe. Only trees with positive and large enough reliability score (larger than median) for each medoid are used and their prediction scores are weighted with the reliability scores. The class with the highest sum of weighted predictions is returned. The described prediction process aims to improve the prediction by using only trees that perform well on instances similar to the given new instance.

To reduce memory consumption of k-medoid algorithm and the space required to store instance similarities, we sample the instances used in the tree reliability estimation process. The size of the sample is a parameter of the method.

7. Evaluation of big data processing in ClowdFlows

To validate our implementation of batch processing of big data we evaluated the big data processing in ClowdFlows by first empirically proving that our map-reduce implementations of algorithms are equivalent in performance to their standard counterparts implemented in widely used libraries. Next we use large data sets and assume that data sets is too big to fit into the main memory of a single worker we test their performance in distributed environment and on assumption that the data set is to big to fit into the main memory of a single worker.

We also validated our newly developed distributed ensemble methods by comparing them to bagging and random forests implemented in the scikit-learn toolkit [44]. Additionally, we test their performance in distributed environment and on assumption that the data set is to big to fit into the main memory of a single worker.

7.1. *Evaluation of DiscoMLL summation form algorithms*

In order to verify the quality of implemented algorithms we compared them with standard single processor based implementations from the scikit-learn toolkit [44]. We used 10 relatively small data sets from UCI repository [80] and 3 big data sets. The characteristics of data sets are presented in Table 3.

We first tested DiscoMLL algorithms based on statistical queries, which, although implemented in a distributed fashion,

Table 3
The characteristics of small UCI data sets (above the line) and big data sets (below the line). The labels in column header have the following meaning: D = number of discrete attributes, R = number of numeric attributes, C = number of class values, N = number of instances, NC = number of chunks used in distributed processing, TC = number of trees per chunk for ensemble methods, Cl = the majority class used in binary classification.

Data set	D	R	C	N	NC	TC	Cl
abalone	1	7	29	2 087	18	30	9
adult	8	6	2	24 420	18	30	$\leq 50K$
car	6	0	4	864	18	30	unacc
isolet	0	617	26	3 899	18	30	17
segmentation	0	19	7	1 155	18	30	sky
semeion	256	0	10	796	18	30	1
spambase	0	57	2	2 300	18	30	–
wilt	0	5	2	2 418	18	30	–
wine-white	0	11	10	2 448	18	30	6
yeast	1	8	10	740	18	30	cyt
covertype	54	0	7	290 506	8	10	2
epsilon	0	2000	2	250 000	65	3	–
mnist8m	0	784	10	4 050 000	119	3	1

follow the same principles as non-distributed implementations and shall achieve the same accuracy.[4]

For MapReduce methods we use a distributed computational environment with 10 nodes (a master node and 9 worker nodes). Each computational node is a 2 CPU AMD Opteron 8431 2.4 GHz with 1 GB RAM using Ubuntu 12.04, so in total we have 18 concurrent processes and each is assigned approximately 1/18 of training instances.

On small data sets 5 × 2 cross-validation was used to test the performance of algorithms. Each training set was randomly split into 18 chunks matching 18 processors in our testing scenario. The results are collected in Table 4. For each algorithm we present two scores: in the left-hand columns the scores of scikit-learn using the whole training set are given and in the right-hand columns the results of DiscoMLL are presented, where each of 18 workers received 1/18 of the training data. We observe that distributed algorithms achieve similar accuracies as non-distributed algorithms. The comparison across algorithms is not possible as for binary classifiers, logistic regression and linear SVM, we binarized all non-binary data sets by setting the label for instances with the most frequent class value to 0 (as indicated in Table 3, column *Cl*) and labels of all the other instances to 1. We also compare the clusterings produced by the k-means algorithm and the clusterings defined by the target labels. The number of clusters for k-means was set to the number of target labels in each data set. In Table 4 we show the values of the adjusted rand index. One can notice that the values for scikit-learn and DMLL are very similar, which indicates that similar clusterings are produced.

In Table 5 we present the results of summation form algorithms on big data sets, for which we assume that they do not fit into the memory. scikit-learn models are therefore trained on subsets with N/NC samples (see Table 3 for these values), while DiscoMLL models are trained on the entire data set distributed over worker nodes (each node contains N/NC samples). All models were tested on the entire test set. The performance of DiscoMLL models is equal or significantly better than the performance of scikit-learn models, except for Naive Bayes and Linear SVM on the mnist8m data set.

7.2. Performance of distributed ensembles

We evaluate the performance of developed ensemble methods (FDDT, DRF and DWF), described in Section 6, and compare their classification accuracy with bagging (*scikit BG*) and random forests (*scikit RF*) implemented in the scikit-learn toolkit. We compare distributed and non-distributed algorithms in two ways:

- giving each worker the whole data set (only possible for small data sets) we expect comparable predictive performance;
- training the distributed ensembles on subsets, we can expect decreased accuracy in comparison with non-distributed methods with all instances at their disposal. Altogether the distributed algorithms can process far more data than single machines so we expect improved performance in comparison to single machines with only chunks of data. For these methods we therefore analyze the decrease of accuracy due to distributed learning. We start with small data sets and observe if the findings generalize to big data sets.

We use a similar testing scenario as for summation form algorithms in Section 7.1, i.e. we use 5 × 2 cross-validation and each training set is randomly split into 18 chunks to match the number of processors in our distributed environment. To measure the decreased performance due to distributed implementations, we simulate the data distribution process using scikit-learn, as follows. We train a model on a subset of 1/18 training instances, predict the entire testing set and measure the classification accuracy. This process is repeated for each of 18 subsets. The same testing method is used with DiscoMLL ensemble algorithms (DiscoMLL subset). We expect similar performance: *scikit subset* $\simeq$ *DiscoMLL subset*. To measure the upper bound of classification accuracy for the algorithms, we train the classification models using all the instances with scikit (scikit ideal). In practice this is not always feasible due to the size of data sets, therefore we also measure the performance in the distributed scenario using the distributed ensembles, which produce models in parallel using subsets and then combine local models in the prediction phase. Due to distributed learning, we expect the following relation between the performance scores: *scikit subset* $\leq$ *DiscoMLL dist* $\leq$ *scikit ideal*.

We present average classification accuracy of distributed and non-distributed ensembles on small data sets in Table 6. To statistically quantify the differences between different methods we test the null hypothesis that distributed ensembles (training a model on the entire data set split into chunks) return models with the same prediction accuracy compared to single processor based implementations, which train models on subsets of data. We applied a paired t-test to measure the significance of differences between matching *scikit subset* and *DiscoMLL dist* scores. In cases when the null hypothesis is rejected we can assume that distributed ensembles are beneficial. The *scikit BG subset* score is compared with *FDDT dist* score and *scikit RF subset* score is compared with *DRF dist* and *DWF dist* scores. The + signs in *FDDT dist*, *DRF dist*, and *DWF dist* columns denote significant improvements of classification accuracy and the - signs denote significant decrease in accuracy. We observe that FDDT achieves significant improvement over scikit BG on every data set except yeast. Similarly DRF achieves a significant improvement over scikit RF on all data sets except on car and wilt data sets. The DWF algorithm achieves several significant improvements over scikit RF, but also two significant decreases of classification accuracy (on wilt and yeast data sets). In general DWF is mostly inferior to DRF. We believe that the reason for this is unreliable assessment of sample similarity based on k-medoid clustering, which is sensitive to noise in the form of less important features. This causes certain samples labeled differently to appear similar. Improvement in the efficient instance similarity assessment is a topic of future work.

Based on the above performance we can confirm our expectations and report a significant increase of accuracy for distributed methods, which use the entire data set split into chunks, compared to single processor methods using only subsets of data. On

[4] In contrast to that, the implemented distributed ensemble methods are not equivalent to the non-distributed implementations as we train them in a distributed fashion using subsets of training data. We compare them to scikit-learn ensemble methods in Section 7.2.

Table 4
Results of summation form algorithms in distributed and non-distributed fashion on small data sets. Classification accuracy is presented for classification algorithms and the adjusted rand index is given for *k*-means clustering. For logistic regression and Linear SVM the data sets are binarized. The + and − signs denote a significant increase or decrease of classification accuracy, respectively.

Data set	Naive Bayes		Logistic regression		Linear SVM		*k*-means	
	scikit	DMLL	scikit	DMLL	scikit	DMLL	scikit	DMLL
abalone	0.22	0.23	0.83	0.83−	0.83	0.83	0.04	0.06+
adult	0.81	0.83+	0.80	0.82+	0.80	0.81+	−0.01	−0.01
car	0.71	0.84+	0.86	0.86	0.87	0.86	0.02	0.01
isolet	0.81	0.81	1.00	0.99−	1.00	0.99−	0.46	0.44
segmentation	0.77	0.77	1.00	1.00+	1.00	1.00	0.37	0.36
semeion	0.82	0.84+	0.97	0.91−	0.97	0.95−	0.37	0.38
spambase	0.82	0.81	0.92	0.92	0.92	0.89−	0.04	0.04
wilt	0.88	0.88	0.95	0.97+	0.95	0.94−	−0.01	−0.01
wine-white	0.44	0.44	0.56	0.56+	0.56	0.56	0.01	0.01
yeast	0.23	0.23	0.70	0.69	0.70	0.70	0.03	0.03

Table 5
Results of summation form algorithms on big data sets. scikit-learn trains models on data subsets while DiscoMLL trains models on the entire data set, but in distributed fashion. Classification accuracy is presented for classification algorithms and adjusted rand index is given for clustering.

Data set	Naive Bayes		Logistic regression		Linear SVM		*k*-means	
	scikit	DMLL	scikit	DMLL	scikit	DMLL	scikit	DMLL
covertype	0.26	0.68	0.76	0.76	0.76	0.76	0.00	0.00
epsilon	0.60	0.67	0.80	0.90	0.80	0.90	0.00	0.00
mnist8m	0.49	0.46	0.98	0.98	0.98	0.95	0.27	0.29

Table 6
Classification accuracy of ensemble methods in distributed and uniprocessor mode on small data sets. The scikit BG subset performance is compared with FDDT dist and scikit RF subset performance is compared with DRF dist and DWF dist. The + and − signs denote a significant improvement and reduction of classification accuracy, respectively.

Data set	scikit BG		scikit RF		FDDT		DRF		DWF	
	subset	ideal	subset	ideal	subset	dist	subset	dist	subset	dist
abalone	0.22	0.23	0.22	0.24	0.22	0.26+	0.22	0.26+	0.22	0.26+
adult	0.84	0.86	0.84	0.85	0.84	0.86+	0.85	0.86+	0.84	0.86+
car	0.79	0.96	0.77	0.95	0.79	0.84+	0.78	0.80	0.77	0.79
isolet	0.75	0.90	0.76	0.94	0.72	0.88+	0.74	0.88+	0.72	0.85+
segmentation	0.87	0.97	0.87	0.97	0.88	0.92+	0.88	0.92+	0.80	0.89+
semeion	0.54	0.84	0.58	0.92	0.48	0.77+	0.53	0.82+	0.50	0.80+
spambase	0.89	0.95	0.91	0.95	0.90	0.92+	0.91	0.92+	0.91	0.93+
wilt	0.96	0.98	0.96	0.98	0.96	0.97+	0.96	0.95	0.96	0.95−
wine-white	0.49	0.64	0.50	0.65	0.51	0.53+	0.51	0.55+	0.49	0.54+
yeast	0.46	0.61	0.43	0.61	0.46	0.51	0.48	0.50+	0.44	0.40−

the other hand, the performance of distributed methods is well beyond the upper bound achieved by using the entire data set in non-distributed mode which leaves much opportunity for further research.

Table 7 presents the results of ensemble methods on big data sets, which are too big to fit into memory of a single machine (ideal scores cannot be computed). For these data sets, scikit-learn algorithms use subsets with N/NC samples while DiscoMLL distributed ensembles use the entire data sets split into chunks to train a model. To make parameters comparable, scikit-learn ensembles construct all the trees on a single subset, while DiscoMLL ensembles construct less trees per subset (as indicated in Table 3, column TC) and combine them into ensembles of equal size. We observe that distributed ensembles mostly achieve higher accuracy. This confirms the benefits of a distributed approach: for data sets too large to fit into memory, the distributed ensemble methods can use more data split into chunks and thereby outperform the non-distributed methods.

Fig. 10 shows learning time of DiscoMLL ensemble methods and their speedup with different number of CPUs on artificially increased [68] segmentation data set (for other data sets the behavior is similar). All methods achieve almost ideal linear speedup i.e. if the time using one node (2 CPUs) is t, the time using x nodes ($2x$ CPUs) is only slightly larger than t/x.

The actual times used by the methods are parameter dependent, but in general DWF is slower than DRF and FDDT as it uses

Table 7
Results of ensemble methods on big data sets, which do not fit into memory of single machines. Scikit algorithms use subsets of data, while DiscoMLL algorithms use distributed computation (dist) to learn on the entire data sets split into chunks.

Data set	scikit BG subset	scikit RF subset	FDDT dist	DRF dist	DWF dist
covertype	0.82	0.79	0.82	0.82	0.82
epsilon	0.71	0.70	0.73	0.74	0.73
mnist8m	0.89	0.92	0.90	0.92	0.92

clustering. DRF is faster than FDDT if it uses the same number of trees as it estimates only $\sqrt{a}$ features in each tree node, where a is the number of attributes. FDDT is faster than scikit RF using the same number of processors.

8. Comparison and integration of ClowdFlows with related platforms

ClowdFlows is an open source data mining platform that can successfully process big data and handle potentially infinite streams of data.

The graphical user interface of ClowdFlows is implemented as a web application that is executed on a remote server. This distinguishes the platform from other platforms such as RapidMiner, KNIME, Weka, and Orange which require an installation which poses distinct software and hardware requirements. The side effect of

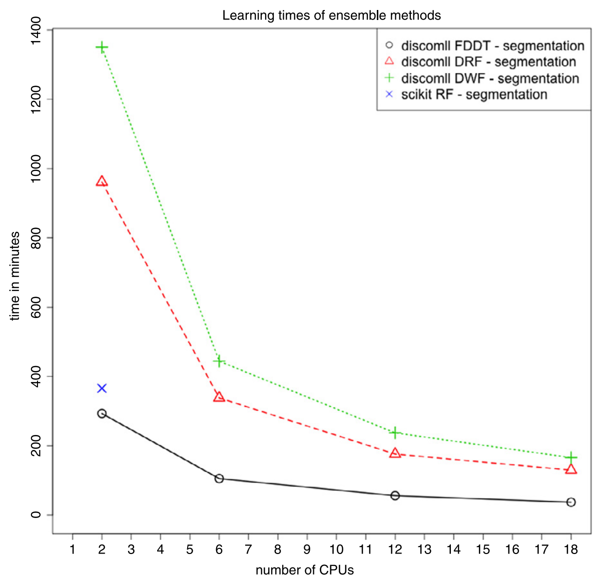


Fig. 10. Learning times of MapReduce ensemble methods with different number of CPUs.

this feature is that users can share their work with anybody as only a web browser is required to view, modify or execute a workflow in ClowdFlows.

Even though ClowdFlows can be deployed on a single machine it is designed with a modular architecture and scalability in mind. Different parts of the system are decoupled so that they may be duplicated for higher performance.

The non-local nature of ClowdFlows makes it an ideal platform for running long term workflows for processing data streams as users do not need to worry about leaving their machines turned on during the process of mining the stream.

ClowdFlows is to the best of our knowledge the only platform that provides a graphical user interface to the Disco Framework and publicly available implementations of data mining algorithms for big data mining in this framework.

The ClowdFlows platform can also be regarded as an open source alternative to the Microsoft Azure Machine Learning platform which requires a subscription with the Azure cloud. ClowdFlows is released under a permissive open source license and can be deployed on public or private clouds.

ClowdFlows in its current state cannot handle processing finished workflows of other platforms but can integrate procedures and algorithms implemented in other platforms and use them internally as part of its own workflows. ClowdFlows features algorithm implementations from Weka, Orange, and scikit-learn. In this way ClowdFlows can be used as a graphical user interface for other platforms, including platforms that do not have graphical user interfaces for constructing workflows, such as TensorFlow. While the TensorFlow interface is not yet implemented in ClowdFlows, it is a subject for further work. The platform is compatible with ClowdFlows as they are both interfaced and expressed in Python.

The ClowdFlows platform can be used within other data mining platforms either by calling ClowdFlows Python functions or by importing REST API services deployed by any live installation of ClowdFlows. The second method exposes an HTTP URL endpoint which executes the workflow when input data is posted to it.

The ClowdFlows platform has been evaluated in independent surveys where it was valued as one of the leading platforms with regards to the number of features [81].

9. Conclusions and further work

We presented the ClowdFlows, a data mining platform that supports the construction and execution of scientific workflows. The platform implements a visual programming paradigm, which 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. With this feature the processing abilities of the ClowdFlows platform are not limited to the initial roster of processing components, but can be expanded with web services. The interface for constructing and monitoring of workflow execution is implemented as a web application and can be accessed from any contemporary web browser. The data and the workflows are stored on the server or a cluster of servers (i.e., a cloud), so that the users are not limited to a single device to access their work. Similarly, workflows are not limited to a single user, but can be made public, which allows other users to use existing workflows either to reproduce the experiments, or as templates to expand and create new workflows.

We presented two modules for big data processing: a real-time analysis module and a batch processing module. Both modules are accessible via an intuitive graphical user interface that are easy to use for data mining practitioners, students, and non-experts.

The stream mining mode uses the stream mining daemon that executes workflows in parallel. The workflow components offer several novel features, so that workflows can connect to potentially infinite number of data streams and process their data. We demonstrated their use by extracting semantical triplets from a live RSS feed.

For analyzing big data in batch mode we developed DiscoMLL, a machine learning library for the Disco MapReduce framework and several ClowdFlows widgets that can interact with a Disco cluster and issue MapReduce tasks. We implemented several summation form algorithms and developed three new ensemble methods based on random forests. Performance analysis shows the benefit of using all available data for learning in the distributed mode compared to using only subsets of data in the non-distributed mode. We demonstrate the ability of our implementation to handle big data sets and its nearly perfect linear speedup.

Both the batch-mode and real-time processing modules were demonstrated with practical use cases that can be reproduced and executed either on the public installation of the ClowdFlows platform, or on a private cluster.

There are several directions for future work. First, the ClowdFlows platform currently implements its own stream processing engine which is built-in and easy to use. On the other hand, the Storm project is widely accepted as a de facto solution for massive stream processing and we plan to provide a loose integration of Storm into the ClowdFlows platform. Likewise, the Apache Hadoop and Spark will be integrated to complement the Disco framework.

Second, the existing ClowdFlows workflow engine will be extended to support different underlying processing platforms. The workflow engine should be able to delegate and monitor tasks transparently providing an easy-to-use programming interface.

Third, the ClowdFlows platform currently is not able to import or export workflows from other visual programming tools. For example, workflows constructed in Taverna or RapidMiner using web services and standard input/output components could easily be imported.

Fourth, we will simplify the installation procedures of ClowdFlows clusters by providing one-click deployment and automatization of scaling.

Finally, the available widget repository will be extended with high quality open-source data processing libraries to cover several new data analysis scenarios, e.g., high-throughput bioinformatics, large scale text processing, graph mining, etc.

To conclude, we believe that ClowdFlows has the potential to become a leading platform for data mining and sharing experiments and results due to its open source nature and its non-opinionated design regarding its collections of workflow components. It is our strategic vision for developers to create their own workflow components, expand the ClowdFlows workflow repository and deploy their own versions of ClowdFlows as a part of a large ClowdFlows network. Since ClowdFlows was released as open source software it has been forked many times and deployed on public servers with custom opinionated sets of workflow components (e.g. ClowdFlows Unistra and TextFlows [82]). With the addition of big data mining and stream mining capabilities presented in this paper we expect the number of users and ClowdFlows installations to increase.

Acknowledgments

This work was supported by the Slovenian Research Agency (Grant Agreement No. P2-0103) and the project ConCreTe, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET Grant No. 611733, and the Human Brain Project (HBP, Grant Agreement No. 604102). We wish to thank Borut Sluban for implementing VIPER in the ClowdFlows platform.

Appendix. Summation form algorithms in DiscoMLL

Besides distributed ensemble methods (FDDT, DRF, and DWF) presented and analyzed in Section 6, DiscoMLL contains implementations of several existing machine learning algorithms in summation form. Since to the best of our knowledge their pseudo code in MapReduce remains unpublished, but is of interest to the scientific community, we include it in this appendix. We present Naive Bayes, logistic regression, K-means clustering, linear regression, locally weighted linear regression, and support vector machine algorithms.

Naive Bayes for numeric attributes

Naive Bayes (NB) for discrete attributes was presented as an example in Section 5.5.1. Here we describe handling of numerical attributes. On numerical features the NB classifier (also called Gaussian Discriminative Analysis in this context) uses numerical features to learn the following statistics: mean, variance and prior probability $P(y)$. The map function (Algorithm 3) takes a training instance, breaks it into individual features and generates output key/value pairs. Each output pair contains the training label y and the feature index j as the key and the feature value x_j as the value. The occurrences of training labels are output in pairs, the training label as the key and 1 as the value. The combiner calculates local statistics (mean, variance and prior probability) for each map task to reduce network load. The reduce function (Algorithm 4) accepts partially calculated statistics for each attribute and combines them appropriately. The statistics are output and used to build a model, which is applied in the predict phase. The output of the predict phase was compared to the Naive Bayes algorithm implemented in the scikit-learn toolkit [44].

We combined the NB classifier for discrete and numeric features into a single algorithm. The computed conditional scores for discrete and numeric attributes are combined into a single score to predict the label $\hat{y}$ with the maximal score as stated in Eq. (A.1) below.

$$\hat{y} = \arg\max_y P(y) \left(\prod_{j \in Numeric} P(x_j|y) \right) \left(\prod_{j \in Discrete} P(x_j|y) \right). \tag{A.1}$$

Algorithm 3: The map function of the fit phase in the NB for numeric attributes.

```
function map(sample, params)
    x, y = sample
    for j = 0 to length(x)
        #key: label, attribute index
        #value: attribute value
        output((y, j), x_j)
    #mark label occurrence
    output(y, 1)
```

Algorithm 4: The reduce function of the fit phase in the NB for numeric attributes.

```
function reduce(iterator, params)
    y_dist = hashmap()
    for key, values in iterator
        if key has 1 element
            #count label occurrences
            y_dist.put(key, sum(values))
        else if key has 2 elements
            #combine local statistics
            mean = calculate_mean(values)
            var = calculate_variance(values)
            output(key, mean)
            output(key, var)
    prior = calculate_prior_probs(y_dist)
    output("prior", prior) #P(y)
```

Logistic regression

The logistic regression classifier is a binary classifier that uses numeric features. The classifier learns by fitting θ to the training data, using the hypothesis in the form $h_\theta(x) = g(\theta^T x) = 1/(1 + \exp(-\theta^T x))$. We use the Newton–Raphson method to update $\theta := \theta - H^{-1}\nabla_\theta \ell(\theta)$. For the summation form, we calculate the subgroups of gradients by map tasks, denoted as $\nabla_\theta \ell(\theta)$, by $\sum_{subgroup}(y - h_\theta(x))x_j$, and the Hessian matrix by $H(j,k) := H(j,k) + h_\theta(x)(h_\theta(x) - 1)\ x_j\ x_k$. The subgroups of the gradient and the Hessian matrix can be computed in parallel by map tasks as shown in Algorithm 5. The logistic regression updates θ in each iteration, where one iteration represents one MapReduce job. Before the execution of each MapReduce job, the θ are stored in object *params* and passed as argument. The map function calculates the hypothesis with x and θ from the previous iteration. The subgroups of gradient and Hessian matrix are calculated and output. The reduce function (Algorithm 6) takes an iterator over key/value pairs. Values with the same key are grouped together by the intermediate phase. The reduce function sums subgroups of gradients and Hessian matrix, updates θ and outputs it. This procedure takes place until convergence or a user-specified number of iterations. The output of the predict phase was compared with the logistic regression algorithm implemented in Orange [11].

Algorithm 5: The map function of the fit phase in the logistic regression.

```
function map(sample, params)
    x, y = sample
    h = calc_hypothesis(x, params.thetas)
    output("grad", calc_gradient(x, y, h))
    output("H", calc_hessian(x, h))
```

Algorithm 6: The reduce function of the fit phase in the logistic regression.

```
function reduce(iterator, params)
    for key, value in iterator
        if key == "H"
            H = sum(value)
        else
            grad = sum(value)
    thetas = params.thetas − inv(H) ∗ grad
    output("thetas", thetas)
```

K-means clustering

The *k*-means is a partitional clustering technique that aims to find a user-specified number of clusters (k) represented by their centroids. The computation of distances between the training instances and centroids can be parallelized. In the initial iteration, the map function randomly assigns data points to k clusters. The mean of data point values, assigned to a certain cluster, defines its centroid. The MapReduce procedure is repeated until it reaches a user-specified number of iterations. The map function (Algorithm 7) takes *sample* as the input parameter, which represents a data point. It computes the Euclidean distance between a data point and each centroid. It assigns each data point to the closest centroid and outputs the cluster identifier as key and the data point as value. The reduce function (Algorithm 8) recomputes centroids for each cluster and outputs the cluster identifier as key and the updated centroid as value. The k reduce tasks are parallelized across the cluster, where each task recalculates a certain centroid. The implementation of the k-means algorithm was taken from Disco examples and was adapted to work with DiscoMLL. The output of the predict phase was compared to the k-means implementation in the scikit-learn toolkit [44].

Algorithm 7: The map function of the fit phase in the k-means.

```
function map(sample, params)
    distances = calc_distances(sample,
                               params.centers)
    center_id = min(distances)
    output(center_id, sample)
```

Algorithm 8: The reduce function of the fit phase in the k-means.

```
function reduce(iterator, params)
    for center_id, samples in iterator
        update_center(params.centers[center_id],
                      samples)

    for center_id, samples in params.centers:
        output(center_id, average(samples))
```

Linear regression

The linear regression fits θ to training data with the equation $\theta^* = A^{-1}b$, where $A = \sum_{i=1}^{m}(x_i x_i^T)$ and $b = \sum_{i=1}^{m}(x_i y_i)$ with m training instances. To put these equations into summation form, the map function calculates $\sum_{subgroup}(x_i x_i^T)$ and $\sum_{subgroup}(x_i y_i)$ as shown in Algorithm 9. The reduce function (Algorithm 10) iterates over subgroups of A and b and sums them. Then it calculates the equation for θ^* and outputs the parameters.

Algorithm 9: The map function of the fit phase in the linear regression.

```
function map(sample, params)
    x, y = sample
    A = outer_product(x, x)
    b = inner_product(x, y)
    output("A", A)
    output("b", b)
```

Algorithm 10: The reduce function of the fit phase in the linear regression.

```
function reduce(iterator, params)
    for key, value in iterator
        if key == "A"
            A = sum(value)
        else:
            b = sum(value)
    thetas = inner_product(inv(A), b)
    output("thetas", thetas)
```

Locally weighted linear regression

Locally weighted linear regression (LOESS) stores training data and computes a linear regression at prediction time, separately for each testing instance. In the linear regression formula instances are weighted with their distances to the testing point, so that points closer to the given testing instance have a strong effect on prediction. This effect is modeled with parameter τ.

LOESS finds a solution of the equation $A\theta = b$, where

$$A = \sum_{i=1}^{m} w^{(i)}(x^{(i)}(x^{(i)})^T),$$

$$b = \sum_{i=1}^{m} w^{(i)}(x^{(i)}y^{(i)}),$$

where $w^{(i)}$ are distance based weights, $x^{(i)}$ is a training instance, and $y^{(i)}$ is function value of instance i. For summary form computation we use map function to compute subsets for A and b, while reduce sums subsets and finds solution to $\theta = A^{-1}b$.

LOESS makes one pass through training data for prediction of a single testing instance, which takes too much time for prediction of many instances. Our implementation computes *theta* parameters for several instances in one pass (Algorithm 11).

Algorithm 11: Computation of θ parameters with LOESS.

```
def get_thetas(train_set, test_set, tau = 1):
    result_url = [], test_set = {}
    read_test_set = read(test_set)
    for id, x in read_test_set:
        test_set[id] = x
        if below_max_capacity(test_set):
            params = Params(test_set, tau)
            # compute thetas for given subset
            thetas = compute(train_set,params)
            results_url.append(thetas)
            testing_set = {}
    return results_url
```

Function (Algorithm 12) uses dictionary of testing instances. Training instances are used to compute weights, and subset of matrices A and b for all testing instances. Function map returns as many pairs as there are testing instances in a dictionary.

Algorithm 12: The map function for LOESS

```
def map(instance, params):
    xi, y = instance
    for id, x in params.test_instances:
        w = weights(xi, x, params.tau)
        sub_A = w * outer_product(xi, xi)
        sub_b = w * xi * y
        yield (id, (sub_A, sub_b))
```

Function reduce (Algorithm 13) sums all subsets of matrices for test instances with the same id. It sorts pairs on the key and merges values based on the key (function *kvgroup*). For each test case we sum subsets of matrices A and b, and compute parameters θ.

Algorithm 13: Function reduce for LOESS.

```
def reduce(iterator, params):
    for id, value in kvgroup(iterator):
        A, b = 0, 0
        for sub_A, sub_b in value:
            A += sub_A
            b += sub_b
        thetas = vector_product(inverse(A),b)
        prediction = vector_product(thetas,
                        params.test_set[id])
        yield (id, (thetas, prediction))
```

Support Vector Machines

We implemented an incremental linear SVM, described in [83], which requires a single pass over the training set with time complexity $O(n^3)$ and space complexity $O(n^2)$. The method assumes numeric attributes, and a binary class encoded with -1 or $+1$. A training set with n instances and m attributes is stored in matrix A, and class values are stored in a diagonal matrix D. We compute matrix $E = [A - e]$, where e is a unit matrix of dimension $m \times 1$. For a user supplied parameter ν, we compute

$$\begin{bmatrix} w \\ \gamma \end{bmatrix} = \left(\frac{I}{\nu} + E^T E\right)^{-1} E^T De. \tag{A.2}$$

With the map function (Algorithm 14) we compute $E^T E$ and $E^T De$ in a distributed fashion.

Algorithm 14: Function map for linear SVM.

```
def map(sample, params):
    A, D = sample
    e = unit_matrix(len(A), 1)
    E = column_merge(A, −e)
    ETE = inner_product($E^T$, $E$)
    ETDe = inner_product($E^T$, D, e)
    yield ("key", (ETE, ETDe))
```

In the reduce function (Algorithm 15) we sum *ETE* and *ETDe*, create the unit matrix I of size $(n+1) \times (n+1)$, which is divided by parameter ν. Using (A.2) we return parameters of linear SVM of size $(n+1) \times 1$.

Algorithm 15: Reduce function of linear SVM.

```
def reduce(iterator, params):
    sum_ETE, sum_ETDe = 0,0
    for key, value in iterator:
        if key == "ETE":
            sum_ETE += value
        else:
            sum_ETDe += value
    I = unit_matrix(sum_ETE.dimension_x)
    sum_ETE += I/params.nu
    yield ("key", vector_product(
            inverse(sum_ETE), sum_ETDe))
```

Each test instance x is extended with 1 in the prediction phase, $z = [{}^{x}_{-1}]$and we get the prediction y as

$$y = sign\left(z^T \left(\frac{I}{\nu} + E^T E\right)^{-1} E^T De\right). \tag{A.3}$$

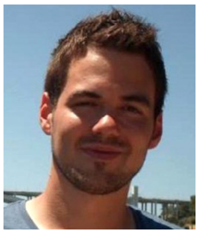
References

[1] J.W. Tukey, The future of data analysis, Ann. Math. Statist. 33 (1) (1962) 1–67.
[2] Q. Yang, X. Wu, 10 challenging problems in data mining research, Int. J. Inf. Technol. Decis. Mak. 05 (04) (2006) 597–604.
[3] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, T. Widener, The KDD process for extracting useful knowledge from volumes of data, Commun. ACM 39 (1996) 27–34.
[4] J. Chambers, Computing with data: Concepts and challenges, Amer. Statist. 53 (1999) 73–84.
[5] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Burlington, Massachusetts, ISBN: 1-55860-238-0, 1993.
[6] J. He, Advances in data mining: History and future, in: Third International Symposium on Intelligent Information Technology Application, IITA 2009, Vol. 1, 2009, pp. 634–636.
[7] I.H. Witten, E. Frank, M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, third ed., Morgan Kaufmann, Amsterdam, ISBN: 978-0-12-374856-0, 2011.
[8] M.M. Burnett, Visual Programming, John Wiley & Sons, Inc., New York, 2001, pp. 275–283.
[9] 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.), KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2006, pp. 935–940.
[10] M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel, KNIME: The Konstanz information miner, in: C. Preisach, H. Burkhardt, L. Schmidt-Thieme, R. Decker (Eds.), GfKl, Studies in Classification, Data Analysis, and Knowledge Organization, Springer, ISBN: 978-3-540-78239-1, 2007, pp. 319–326.
[11] J. Demšar, B. Zupan, G. Leban, T. Curk, Orange: From experimental machine learning to interactive data mining, in: J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), PKDD, in: Lecture Notes in Computer Science, vol. 3202, Springer, ISBN: 3-540-23108-0, 2004, pp. 537–539.
[12] J. Kranjc, V. Podpečan, N. Lavrač, ClowdFlows: A cloud based scientific workflow platform, in: P.A. Flach, T.D. Bie, N. Cristianini (Eds.), Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, vol. 7524, Springer, ISBN: 978-3-642-33485-6, 2012, pp. 816–819.
[13] Executable Paper Challenge, 2016. URL http://www.executablepapers.com/ (accessed: 18.01.16).
[14] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM (ISSN: 0001-0782) 51 (1) (2008) 107–113. http://dx.doi.org/10.1145/1327452.1327492, URL http://doi.acm.org/10.1145/1327452.1327492.
[15] J. Kranjc, V. Podpečan, N. Lavrač, ClowdFlows sources, 2016. URL http://mloss.org/software/view/513/ (accessed: 20.01.2016).

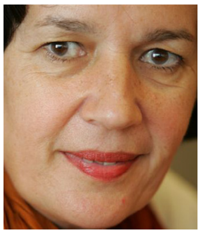
[16] D. Talia, P. Trunfio, O. Verta, Weka4WS: A WSRF-enabled Weka toolkit for distributed data mining on grids, in: A. Jorge, L. Torgo, P. Brazdil, R. Camacho, J. Gama (Eds.), PKDD, in: Lecture Notes in Computer Science, vol. 3721, Springer, 2005, pp. 309–320. ISSN 3-540-29244-6.
[17] V. Podpečan, M. Zemenova, N. Lavrač, Orange4WS environment for service-oriented data mining, Comput. J. 55 (1) (2012) 89–98.
[18] 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.
[19] G. Decker, H. Overdick, M. Weske, Oryx - An open modeling platform for the BPM community, in: M. Dumas, M. Reichert, M.-C. Shan (Eds.), Business Process Management, in: Lecture Notes in Computer Science, vol. 5240, Springer, Berlin/ Heidelberg, ISBN: 978-3-540-85757-0, 2008, pp. 382–385.
[20] D. Blankenberg, G.V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, J. Taylor, Galaxy: A web-based genome analysis tool for experimentalists, in: Frederick M. Ausubel, et al. (Eds.), Current Protocols in Molecular Biology, 2010, ISSN1934-3647 (Chapter 19).
[21] R. Rak, A. Rowley, W. Black, S. Ananiadou, Argo: an integrative, interactive, text mining-based workbench supporting curation, Database: J. Biol. Databases Curation (2012).
[22] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R.F. da Silva, M. Livny, K. Wenger, Pegasus, a workflow management system for science automation, Future Gener. Comput. Syst. (ISSN: 0167-739X) 46 (2015).
[23] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger, Workflow management in condor, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for E-Science, Springer, London, ISBN: 978-1-84628-519-6, 2007, pp. 357–375.
[24] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, M. Wieczorek, ASKALON: A development and grid computing environment for scientific workflows, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for E-Science, Springer, London, ISBN: 978-1-84628-519-6, 2007, pp. 450–471.
[25] E. Elmroth, F. Hernandez, J. Tordsson, Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment, Future Gener. Comput. Syst. (ISSN: 0167-739X) 26 (2) (2010) 245–256. URL http://www.sciencedirect.com/science/article/pii/S0167739X09001174.
[26] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, Kepler: an extensible system for design and execution of scientific workflows, in: Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004, 2004. pp. 423–424, http://dx.doi.org/10.1109/SSDM.2004.1311241.
[27] I. Taylor, M. Shields, I. Wang, A. Harrison, The Triana Workflow Environment: Architecture and applications, in: Workflows for E-Science, 1, 2007, pp. 320–339.
[28] T. White, Hadoop: the Definitive Guide, O'Reilly, 2012.
[29] T. Condie, N. Conway, P. Alvaro, J.M. Hellerstein, K. Elmeleegy, R. Sears, MapReduce online, in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 21–21. URL http://dl.acm.org/citation.cfm?id=1855711.1855732.
[30] C.T. Chu, S.K. Kim, Y.A. Lin, Y. Yu, G.R. Bradski, A.Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, in: B. Schölkopf, J.C. Platt, T. Hoffman (Eds.), NIPS, MIT Press, 2006, pp. 281–288. URL http://www.cs.stanford.edu/people/ang//papers/nips06-mapreducemulticore.pdf.
[31] Apache Mahout: Scalable machine learning and data mining, 2016. URL http://mahout.apache.org/ (accessed: 20.01.16).
[32] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Vol. 10, 10, 2010.
[33] Z. Prekopcsák, G. Makrai, T. Henk, C. Gáspár-Papanek, Radoop: Analyzing big data with rapidminer and hadoop, in: Proceedings of the 2nd RapidMiner Community Meeting and Conference, RCOMM 2011, 2011.
[34] P. Mundkur, V. Tuulos, J. Flatow, Disco: a computing platform for large-scale data analytics, in: Proceedings of the 10th ACM SIGPLAN Workshop on Erlang, ACM, 2011, pp. 84–89.
[35] L. Neumeyer, B. Robbins, A. Nair, A. Kesari, S4: Distributed stream computing platform, in: 2010 IEEE International Conference on Data Mining Workshops, (ICDMW), IEEE, 2010, pp. 170–177.
[36] N. Marz, Storm, distributed and fault-tolerant realtime computation, 2016. URL http://storm-project.net/ (accessed: 20.01.16).
[37] G.D.F. Morales, SAMOA: a platform for mining big data streams, in: L. Carr, A.H.F. Laender, B.F. Lóscio, I. King, M. Fontoura, D. Vrandecic, L. Aroyo, J.P.M. de Oliveira, F. Lima, E. Wilde (Eds.), WWW (Companion Volume), International World Wide Web Conferences Steering Committee /, ACM, ISBN: 978-1-4503-2038-2, 2013, pp. 777–778.
[38] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604.
[39] P. Reutemann, J. Vanschoren, Scientific workflow management with ADAMS, in: P.A. Flach, T.D. Bie, N. Cristianini (Eds.), ECML/PKDD (2), in: Lecture Notes in Computer Science, vol. 7524, Springer, ISBN: 978-3-642-33485-6, 2012, pp. 833–837.
[40] J. Vanschoren, H. Blockeel, A community-based platform for machine learning experimentation, in: W. Buntine, M. Grobelnik, D. Mladenič, J. Shawe-Taylor (Eds.), Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, vol. 5782, Springer Berlin, Heidelberg, ISBN: 978-3-642-04173-0, 2009, pp. 750–754.
[41] C. Kiourt, D. Kalles, A platform for large-scale game-playing multi-agent systems on a high performance computing infrastructure, Multiagent Grid Syst. 12 (1) (2016) 35–54.
[42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow. org.
[43] Microsoft, Microsoft Azure Machine Learning, 2016. URL https://azure.microsoft.com/en-us/services/machine-learning/ (accessed: 04.04.16).
[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[45] JPype - Java to Python Integration, 2016. URL http://jpype.sourceforge.net/ (accessed 20.01.16).
[46] M. Lichman, UCI Machine Learning Repository, 2013. URL http://archive.ics.uci.edu/ml.
[47] B. Sluban, D. Gamberger, N. Lavrač, Ensemble-based noise detection: noise ranking and visual performance evaluation, Data Min. Knowl. Discov. (2013) 1–39.
[48] J. Kranjc, V. Podpecan, N. Lavrac, Real-time data analysis in ClowdFlows, in: 2013 IEEE International Conference on Big Data, IEEE, 2013, pp. 15–22.
[49] jQuery, 2016. URL http://jquery.com/ (accessed 20.01.16).
[50] Usage Statistics and Market Share of JavaScript Libraries for Websites, January 2014, 2016. URL http://w3techs.com/technologies/overview/javascript_library/all (accessed 20.01.16).
[51] Django: The Web framework for perfectionists with deadlines, 2016. URL https://www.djangoproject.com/ (accessed 20.01.16).
[52] A. Richardson, Introduction to RabbitMQ, Google UK, Sep 25.
[53] Python Simple SOAP library, 2016. URL https://code.google.com/p/pysimplesoap (accessed 20.01.16).
[54] E. Ikonomovska, S. Loskovska, D. Gjorgjevik, A survey of stream data mining, in: Proceedings of 8th National Conference with International Participation, ETAI, 2007, pp. 19–21.
[55] D. Rusu, B. Fortuna, M. Grobelnik, D. Mladenič, Semantic graphs derived from triplets with application in document summarization, Informatica (Slovenia) 33 (3) (2009) 357–362.
[56] PyTeaser source code, 2016. URL https://github.com/xiaoxu193/PyTeaser (accessed: 20.01.16).
[57] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, D. Mladenic, Triplet extraction from sentences, in: Proceedings of the 10th International Multiconference, Information Society-IS, 2007, pp. 8–12.
[58] Stanford Parser, 2016. URL http://nlp.stanford.edu/software/lex-parser.shtml (accessed: 20.01.16).
[59] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python, O'Reilly Media Inc., 2009.
[60] G.A. Miller, WordNet: a lexical database for English, Commun. ACM 38 (11) (1995) 39–41.
[61] M. Bostock, V. Ogievetsky, J. Heer, D^3 data-driven documents, IEEE Trans. Vis. Comput. Graphics 17 (12) (2011) 2301–2309.
[62] T. Dwyer, Scalable, versatile and simple constrained graph layout, in: Computer Graphics Forum, Vol. 28, Wiley Online Library, 2009, pp. 991–998.
[63] R. Orač, Disco Machine Learning Library, 2015. URL https://github.com/romanorac/discomll (accessed: 10.01.15).
[64] T.E. Oliphant, Python for scientific computing, Comput. Sci. Eng. 9 (3) (2007) 10–20.
[65] C. Chu, S.K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A.Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, Adv. Neural Inf. Process. Syst. 19 (2007) 281.
[66] M. Kearns, Efficient noise-tolerant learning from statistical queries, J. ACM 45 (6) (1998) 983–1006.
[67] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, ACM, 2006, pp. 161–168.
[68] M. Robnik-Šikonja, Data generators for learning systems based on RBF networks, IEEE Trans. Neural Netw. Learn. Syst. 27 (5) (2016) 926–938.
[69] L. Breiman, Random forests, Mach. Learn. J. 45 (2001) 5–32.
[70] A. Verikas, A. Gelzinis, M. Bacauskiene, Mining data with random forests: A survey and results of new tests, Pattern Recognit. 44 (2) (2011) 330–349.
[71] G. Wu, H. Li, X. Hu, Y. Bi, J. Zhang, X. Wu, MReC4.5: C4.5 ensemble classification with MapReduce, in: Fourth ChinaGrid Annual Conference, IEEE, 2009, pp. 249–255.
[72] J. Basilico, M. Munson, T. Kolda, K. Dixon, W. Kegelmeyer, COMET: A recipe for learning and using large ensembles on massive data, in: 2011 IEEE 11th International Conference on Data Mining, (ICDM), IEEE, 2011, pp. 41–50.
[73] B. Panda, J. Herbach, S. Basu, R. Bayardo, PLANET: Massively parallel learning of tree ensembles with MapReduce, Proc. VLDB Endow. 2 (2) (2009) 1426–1437.
[74] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, 1993.
[75] L. Breiman, Pasting small votes for classification in large databases and on-line, Mach. Learn. 36 (1–2) (1999) 85–103.
[76] I. Kononenko, On biases in estimating multi-valued attributes, in: Proceedings of the International Joint Conference on Artificial Intelligence, (IJCAI'95), Morgan Kaufmann, 1995, pp. 1034–1040.
[77] L. Breiman, Bagging predictors, Mach. Learn. J. 26 (2) (1996) 123–140.
[78] M. Robnik-Šikonja, Improving Random Forests, in: Machine Learning: Proceedings of ECML 2004, 2004.

[79] J. Gower, A general coefficient of similarity and some of its properties, Biometrics (1971) 857–871.
[80] A. Frank, A. Asuncion, UCI Machine Learning Repository, 2010. URL http://archive.ics.uci.edu/ml.
[81] O. Kurasova, V. Marcinkevičius, V. Medvedev, A. Rapečka, Data mining systems based on web services, Informacijos mokslai 65 (2013) 66–74.
[82] M. Perovšek, J. Kranjc, T. Erjavec, B. Cestnik, N. Lavrač, TextFlows: A visual programming platform for text mining and natural language processing, Spec. Issue Knowl.-Based Softw. Eng. Sci. Comput. Program. 121 (2016) 128–152.
[83] G. Fung, O.L. Mangasarian, Incremental support vector machine classification, in: SDM, SIAM, 2002, pp. 247–260.

Janez Kranjc started his graduate studies at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. He is enrolled in the Ph.D. programme "Information and Communication Technologies" and works as a research assistant at the Department of Knowledge Technologies, Jožef Stefan Institute. His research in the field of computer science mostly focuses on developing novel cloud-based approaches to data mining and knowledge discovery using graphical scientific workflows and developing applications and infrastructure to facilitate continuous mining of data streams and vast amounts of data dispersed on the cloud. The applications of the developed platform and workflows span several fields such as data mining, text mining, natural language processing, and sentiment analysis.


Roman Orač completed his B.Sc. in computer science in 2011 and his M.Sc. in 2014. He is a big data enthusiast who finds interest in the field of large-scale data mining. He is currently employed as an analytics engineer at Iddiction.

Vid Podpečan, Ph. D. is a researcher at the Department of Knowledge technologies at the Jožef Stefan Institute, Ljubljana, Slovenia. He received his Ph.D degree in computer science in 2013. His research interests lie mostly within robotics, artificial intelligence and data analysis, with a focus on humanoid robots and their applications, computational creativity, and data mining topics such as network analysis and bioinformatics. His research work has resulted in 45+ scientific publications and several software tools (e.g., Orange4WS, SegMine). Recently, he became involved in promoting robotics with Aldebaran's NAO humanoid robot to younger generations, especially children.


Nada Lavrač is Head of Department of Knowledge Technologies of Jozef Stefan Institute, Ljubljana, Slovenia. She is Professor at Jozef Stefan International Postgraduate School in Ljubljana and at University of Nova Gorica. She was initiator and co-founder of Centre for Knowledge Transfer in Information Technologies at JSI. She is Deputy Head of Information and Communication Technologies M.Sc. and Ph.D. Programs at Jozef Stefan International Postgraduate School in Ljubljana. Her main research interests are in Knowledge Technologies, an area of Information and Communication Technologies. Her particular research interests are machine learning, data mining, text mining, knowledge management and computational creativity.

Marko Robnik-Šikonja received his Ph.D. in computer science in 2001 from the University of Ljubljana. He is an Associate Professor at the University of Ljubljana, Faculty of Computer and Information Science. His research interests include machine learning, data and text mining, knowledge discovery, cognitive modeling, and their practical applications. He is a (co)author of more than 60 publications in scientific journals and international conferences and three open-source analytic tools.