Real-Time Data Analysis in ClowdFlows

Janez Kranjc$^{1,2}$
Vid Podpečan$^1$
$^1$Jožef Stefan Institute
$^2$International Postgraduate School Jožef Stefan
Jamova 39, 1000 Ljubljana, Slovenia
janez.kranjc@ijs.si

Nada Lavrač$^{1,3}$
$^3$University of Nova Gorica
Vipavska 13, 5000 Nova Gorica, Slovenia

Abstract—ClowdFlows is an open cloud based platform for composition, execution, and sharing of interactive data mining workflows. In this paper we extend the ClowdFlows platform with the ability to mine real-time data streams. This functionality was implemented by creating a specialized type of workflow component and a stream mining daemon that delegates the execution of workflows in real-time. In this way, we have transformed a batch data processing platform into a real-time stream mining platform with an intuitive user interface. The real-time analytics aspect of the platform is demonstrated in a Twitter sentiment analysis use case where the sentiment of tweets about whistleblower Edward Snowden was monitored for approximately one month.

Keywords—data mining platform; stream mining; real-time data analysis; web application; workflows; sentiment analysis

I. INTRODUCTION

During recent years performing data mining tasks has become synonymous with constructing data mining workflows in specialized knowledge discovery and data mining environments. In contrast to traditional data analysis software which supported a single or few algorithms designed to mine specialized data, current knowledge discovery systems provide large collections of generic algorithm implementations designed to mine specialized data, current knowledge discovery systems provide large collections of generic algorithm implementations coupled with an easy-to-use graphical user interface. The common feature of all these data mining tools is the concept of visual programming: all advanced knowledge discovery systems offer some form of workflow construction and execution engine. This allows practitioners and non-experts to perform data mining tasks by using simple drag and drop operations on workflow components and connect them to form workflows. Performing real-time data analysis on complex and geographically dispersed information and knowledge sources can also be simplified by providing practitioners with simple ways to construct real-time data mining workflows using a specialized data mining platform.

ClowdFlows$^1$ is a new data mining platform designed to overcome several deficiencies of similar data mining platforms and to provide novel features to benefit the data mining community by allowing mining of big data in real-time by connecting to different data streams. This paper presents the process of transforming the ClowdFlows platform to a real-time data mining platform. The platform was first developed as a traditional data mining tool for processing static data, which successfully bridges different operating systems and platforms, and is able to fully utilize available server resources in order to relieve the client from heavy-duty processing and data transfer as the platform is entirely Web based and can be accessed from any modern browser [1]. ClowdFlows also benefits from a service-oriented architecture which allows users to utilize arbitrary Web services as workflow components.

The paper is structured as follows. Section II presents comparable data mining and stream mining platforms and describes their differences and similarities with ClowdFlows. The technical background and implementation of the platform is presented in Section III. Section IV presents the process of adapting ClowdFlows to work on real-time data streams. In Section V we present a use case for mining the Twitter sentiment on an arbitrary query in real-time. Finally, in Section VI we conclude the paper with ideas for further work.

II. RELATED WORK

This section describes similar data mining platforms, some of which also support stream mining. The platforms are grouped together based on the features they offer. The important features of these platforms are: the implementation of the visual programming paradigm, service-oriented approach to knowledge discovery, remote workflow execution, workflow sharing, and functionalities that allow real-time data analysis. The differences and similarities between ClowdFlows and these platforms are also noted and explained. The related work is separated in two categories, based on whether it focuses on batch or stream processing.

A. Data mining tools and batch processing

The majority of data mining software solutions from different domains allow construction and execution of scientific workflows. Known examples of such platforms include Weka [2], Orange [3], KNIME [4], and RapidMiner [5] data

$^1$http://clowdflows.org/
mining environments. These platforms all implement a workflow canvas which is a part of the graphical user interface where workflows can be constructed by using drag, drop and connect operations on the available components. The workflow components (parts of the graphical user interface) are often referred to as widgets. The range of available components typically includes data loading and pre-processing, data and pattern mining algorithms and interactive and non-interactive visualizations. The graphical user interfaces of these platforms are developed in Java (with the exception of Orange which is written in Python), and need to be installed on the user’s computer. Since ClowdFlows is a Web based platform, the graphical user interface is written entirely in HTML and JavaScript, which allows it to be accessed from a Web browser.

In order to benefit from service-oriented architecture concepts, software tools have emerged, which are able to make use of Web services, and can access large public databases (some supporting means of grid deployment and P2P computing). Environments such as Weka4WS [6], Orange4WS [7], Web Extension for RapidMiner, Triana [8], Taverna [9] and Kepler [10] allow for the integration of Web services as workflow components. However, with the exception of Orange4WS and Web Extension for RapidMiner, these environments are mostly specialized in domains like systems biology, chemistry, medical imaging, ecology and geology.

Remote workflow execution (on different machines than the one used for workflow construction) is employed by KNIME and RapidMiner using the RapidAnalytics server. This allows the execution of 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 still used for designing workflows which are executed on remote machines, while only the results can be viewed using a Web interface.

Sharing data and experiments has been implemented in the Experiment Database [11], 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. Sharing of workflows has also been implemented at the myExperiment website [12]. Users can save workflows from several data mining platforms and upload them to the myExperiment website. Using the myExperiment website as a medium for sharing workflows still requires installation of an appropriate data mining platform on the user’s computer. ClowdFlows supports sharing data and workflows natively by providing users with special URLs for their workflows that can be accessed from anywhere and by anyone (if permitted by the author of the workflow).

B. Data mining tools and stream processing

Stream classification and clustering was implemented in the MOA framework [13]. MOA is related to Weka and also written in Java. The MOA framework however cannot process streams in a distributed environment.

Distributed Stream Processing Engines have been the focus of much research and development recently. One of the first open source Stream Processing Engines available was Borealis [14]. The SAMOA project is an upcoming project that will provide the MOA functionalities using the S4 [15] and Storm SPEs to create a distributed stream mining platform. ClowdFlows implements its own distributed Stream Processing Engine which is based on its own workflow execution engine used for batch processing. The engine is simpler in comparison to Storm or S4, but easy to install and requires no programming skills to use, due to its graphical user interface.

III. THE CLOWDFLOWS PLATFORM

The development of the ClowdFlows platform is presented in this section. The enabling technologies are described briefly and displayed in the architecture of the system. The graphical user interface and the basic usage of ClowdFlows are presented along with its workflow execution engine.

A. Architecture of the platform and technologies used

The ClowdFlows platform is first and foremost a Web application. It consists of mainly two parts: the server-side and the client-side. The server side is the part of the application that is executed on a server or a cluster of servers. The client side present the parts of the system that are executed on the user’s computer. The architecture of the system being accessed by multiple users is illustrated in Figure 1.

The graphical user interface is implemented in HTML and CSS, so that it may be rendered in Web browsers. The functionalities of the user interface were implemented in JavaScript using the jQuery library. The graphical user interface is accessed with a Web browser. A public installation of ClowdFlows is available at the following URL: http://clowdflows.org.

The server side is written in Python using the Django framework. Python was selected because it allowed easy integration of many machine learning and data mining tools implemented in Python, such as Orange [3] and Scikit-learn [16]. The Django framework is a high-level Python Web framework that encourages rapid development and clean, pragmatic design, and is the most popular Web framework for Python. It provides an object-relational mapper and a powerful template system. The object-relational mapper

1http://samoa-project.net/
2http://storm-project.net/
3http://jquery.com/
4https://www.djangoproject.com/
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 all supported, although MySQL is used in the public installation of ClowdFlows.

ClowdFlows may also be installed on multiple computers, which is enabled by using the RabbitMQ\(^6\) messaging server and broker and a Django implementation of Celery\(^7\), a distributed task queue, which allows passing asynchronous tasks between servers. Workers that execute tasks can be installed on a cluster of machines, or on a single machine.

To enable consumption of Web services and using them as workflow components, the PySimpleSoap library\(^8\) is used. PySimpleSoap provides an interface for client and server Web service communication, which allows importing WSDL

\(^6\)http://www.rabbitmq.com/
\(^7\)http://celeryproject.com/
\(^8\)https://code.google.com/p/pysimplesoap/

Web services as workflow components, and exposing entire workflows as WSDL Web services.

B. The Graphical User Interface

The graphical user interface follows the visual programming paradigm for constructing workflows. This simplifies the representation of complex procedures into a spatial arrangement of building blocks. The building blocks or workflow components in ClowdFlows are called widgets. Each building block receives some data or parameters at the input, performs a specific task, and return the results on its outputs. Workflows consist of widgets and connections that connect outputs of widgets to inputs of other widgets.

The graphical user interface of the ClowdFlows platform consists of two main parts: the widget repository and the workflow canvas. A screenshot of the system can be seen in Figure 2. The widget repository is located on the left side of the screen and the workflow canvas on the right. The workflow canvas is in the screenshot features a trivial workflow for building and displaying a J48 decision tree.

The widget repository is a taxonomy of workflow components that can be put on the canvas by clicking on them. Widgets in ClowdFlows belong into four basic groups. Regular widgets are widgets that simply take data at the input, perform a specific task and return data on their outputs. Visualization widgets can render results as HTML and Javascript and display it in dialog windows in the browser. Interactive widgets can open a modal window to query the user for additional data during run-time. Finally, workflow control widgets allow creating sub-workflows (i.e. a widget that contains a workflow) and for loops. The widgets repository contains widgets from several different data mining platforms. Orange widgets were implemented into ClowdFlows as they are both written in Python. Weka algorithms were exposed as Web services and imported into the platform. The scikit-learn machine learning library in Python was also implemented as a set of widgets.
The workflow canvas implements moving, connecting, deleting and issuing commands to execute widgets and workflows. Widgets can be arbitrarily arranged on the canvas by dragging and dropping. Connections between widgets are created by selecting an output of a widget and an input of another widget. Whenever a user performs an action on the graphical user interface it is sent to the server using an asynchronous HTTP POST request. Operations are validated on the server and the success or error messages are passed to the client’s browser formatted in JavaScript Object Notation (JSON) or HTML.

A toolbar is located above the workflow canvas and can be used to save, copy, and delete entire workflows. When a workflow is saved it can also be made public. Public workflows get specialized URLs, which can be shared so that other people can access these workflows.

C. The Workflow Engine

The workflow execution engine executes all executable widgets in the workflow in the correct order. The engine operates with widgets and assigns them different states based on their state of execution.

A state of each widget can be one of the following: executable, pending, running, finished, or failed. The goal of the workflow execution engine is to have no widgets in the executable or running state in the workflow.

Executable widgets are widgets whose predecessors’ state is finished, or have no predecessors at all. A predecessor of a widget is a widget whose output is connected to an input of its descendant. Pending widgets are widgets whose state is neither executable, running, finished nor failed. Running widgets are widgets that have been executed but have not yet finished or failed. Finished and failed widgets are widgets that completed the execution either successfully or unsuccessfully. A failed or finished widget becomes pending or executable whenever a connection leading towards the widget is deleted, added, or changed, or when a value of this widget’s parameter is changed, or when a user manually changes the state of the widget to executable.

When a workflow is running the execution engine perpetually checks for widgets that are executable and executes them. Whenever two or more widgets are executable at the same time they are asynchronously executed in parallel, since they are independent. The widget state mechanism ensures that no two widgets where the inputs of a widget are dependent on an output of another widget will be executable at the same time. The execution of a workflow is complete when there are no executable or running widgets.

IV. Real-Time Data Analysis in CloudFlows

This section describes the adaptation of CloudFlows to enable real-time data processing. The workflow execution engine has been augmented with continuous parallel execution and the halting mechanism. Specialized widgets that are useful in real-time processing workflows are also described in this section.

A. Continuous workflow execution with the halting mechanism

Regular workflows and stream mining workflows are primarily distinguished by their execution times. A widget in a static workflow is executed a finite amount of times and the workflow has a finite execution time. Widgets in a stream mining workflow are executed a potentially infinite amount of times and the workflows are executed until manually terminated by users. Another major difference between regular workflows and stream mining workflows is the data on the input. The data that is processed by regular workflows is available in whole during the entire processing time, while data entering the stream mining workflows is potentially infinite and is only exposed as a small instance at any given time.

In order to handle potentially infinite data streams we have modified the workflow execution engine to execute the workflow multiple times at arbitrarily small temporal intervals in parallel. The amount of parallelism and the frequency of the execution are parameters that can be (providing the hardware availability) modified for each stream to maximize the throughput.

The execution of the workflows is delegated by a special stream mining daemon that issues tasks to the messaging queue. The stream mining daemon’s task is to issue commands to execute streaming workflows. The daemon can also prioritize execution of some streams over others based on the users’ preferences. Tasks are picked up from the messaging queue by workers that execute the workflow. To ensure that each execution of a workflow processes a different instance of the data, special widgets and mechanisms were developed, which can halt the execution of streaming workflows. This halting mechanism can be activated by widgets in a streaming workflow to halt the current execution.

Workflows that are executed as a stream mining process need to be saved as streaming workflows and executed separately. The user cannot inspect the execution of the workflow in real time, as many processes are running in parallel. The user can, however, see results from special stream visualization widgets.

B. Specialized workflow widgets for real-time processing

Widgets in stream mining workflows have, in contrast to widgets in regular workflows, internal memory and the ability to halt the execution of the current workflow. 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 had to be 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 just the timestamp of the latest instance if they are available in the stream itself. If the input widget encounters no new data instances at the stream source it halts the execution of the stream. No other widgets that are directly connected to it via its outputs will be executed until the workflow is executed again.

Other popular stream mining approaches [17] were also 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 the 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 has been reached.

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 are fairly simple. They either pass the instance to the next widget or halt the execution, based on an arbitrary condition. This condition can be dependent on the data or not (e.g. drop every second instance). The internal memory can be used to store counters, which are used to decide which data is left in the sample.

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

V. TWEET SENTIMENT ANALYSIS USE CASE

In this section we present a use case that showcases some of the features of the ClowdFlows real-time data processing platform. The description of the use case is written as a step-by-step report on how the workflow was constructed. Following the description in this section, it is possible for the reader to construct a fully functioning stream mining workflow and view the results.

The aim of the use case is to monitor the Twitter sentiment on a given subject. For the purpose of this use case we have selected to monitor tweets containing the keyword Snowden, as it is one of the trending keywords during the time of writing this article. We wish to measure the Twitter sentiment over time regarding Edward Snowden, who leaked details of several top-secret documents to the press.

A. The development of necessary components

To construct the workflow we required a stream input that would collect tweets based on a query, a sampling widget that would discard any non-English tweets, a widget to perform sentiment analysis on tweets, a stream splitter to split the stream of tweets into a stream of positive and a stream of negative tweets, and three types of visualization widgets to display the line chart of the sentiment over time, a word cloud of positive or negative tweets, and the latest tweets.

To consume the incoming stream we implemented a widget that connects to Twitter via the Twitter API\(^9\). The widget accepts several parameters: the search query, by which it filters the incoming tweets, the geographical location (optional), which filters tweets based on location, and the credentials for the Twitter API. The widget works both in a streaming and non-streaming environment. Whenever the widget is executed it will fetch the latest results of the search query. For streaming workflows, the internal memory of the widget holds the ID of the latest tweet, which is passed to the Twitter API, so that only the tweets that have not yet been seen are fetched.

Since tweets returned by the Twitter API are annotated with their language, we constructed a widget for filtering tweets based on their language. This widget discards all tweets that are not in English.

There are three common approaches to perform sentiment classification [18]: machine learning based, lexicon based and linguistic analysis. In our case we used an SVM classifier (machine learning based) that was trained on a labelled dataset and implemented in the .NET framework [19]. In order to use it in the CloudFlows platform we exposed the classifier as a Web service and imported it in the platform.

A simple widget was implemented that splits the stream of tweets into two streams, based on their sentiment. This was done so that positive and negative tweets could be separately inspected.

To visualize the sentiment we implemented a line chart that displays the volume of all tweets, volume of positive tweets, negative volume of negative tweets, and the difference of positive and negative tweets. The visualization was implemented with the HighCharts JavaScript visualization library\(^10\).

To inspect separate tweets a simple table was implemented where each tweet is coloured red or green based on its sentiment (red for negative and green for positive).

The word cloud visualization was implemented to show most popular words in recent tweets. This visualization is

\(^9\)https://dev.twitter.com/

\(^10\)http://www.highcharts.com/
dynamic and changes with the stream. By looking at the word cloud and see popular words appearing and unpopular words disappearing is a novel way to look at data streams in real-time. The visualization was developed with the D3.js JavaScript library.\textsuperscript{11}

\textbf{B. Constructing the workflow}

The workflow was constructed using the ClowdFlows graphical user interface. Widgets were selected from the widget repository and added to the canvas and connected as shown in Figure 3.

After connecting the widget into the workflow, some parameters were set. Parameters of a widget are set by double clicking the widget. Twitter API credentials and the search query were entered as parameters for the Twitter widget. The language code \textit{en} was entered as a parameter of the Filter tweets by language widget. We have also added three sliding window widgets with the size 500 (entered as parameter) to the workflow. This is done because the visualization widgets that display tweets and word clouds only display the last data that was received as an input for these widgets. By setting the size of the window to 500 the word cloud will always consist of the most recent 500 tweets.

The workflow was saved by clicking the save button in the toolbar. We have also marked the workflow as public so that the workflow can be viewed and copied by other people. The URL of the workflow is http://clowdflows.org/workflow/1041/. We have then navigated to the workflows page (http://clowdflows.org/your-workflows/) and clicked the button "Start stream mining" next to our saved workflow. By doing this we have instructed the platform to start executing the workflow with the stream mining daemon. A special web page was created where detailed information about the stream mining process is displayed. This page also contains links to visualization pages that were generated by the widgets. The stream mining process was left running from the 14th of June until the 10th of July.

\textbf{C. Monitoring the results}

We have put several stream visualization widgets in the workflow which allowed us to inspect the results during the process of stream mining. ClowdFlows has generated a Web page for each stream visualization widget, which can be viewed by anybody since the workflow is public.

The sentiment graph visualization displaying the line chart of volumes of tweets, volumes of positive tweets, negative volume of negative tweets and the difference of positive and negative sentiment is available at http://clowdflows.org/streams/data/4/9056/ and is displayed in Figure 4. By looking at this visualization we can see that the general sentiment in the tweets mentioning Snowden is generally more positive than negative. We can observe several spikes in volume which correspond to the times when news articles regarding this subject were published. On June 23rd, news of Edward Snowden’s departure from Hong Kong and arrival in Moscow was published. On the first of July Edward Snowden released a statement on the Wikileaks website and lots of news reports focused on possible countries that could offer asylum to Edward Snowden.

\textsuperscript{11}http://d3js.org/
The word cloud visualization of negative tweets is available at http://clowdflows.org/streams/data/4/9065/ and is shown in Figure 5. This visualization helps put the stream into another perspective and can display changing trends in real-time. When the word cloud is opened in the browser and the stream mining process is active the words change positions and sizes corresponding to their occurrences in the tweets.

Links to the visualizations of other stream visualization widgets are also present on the two provided visualization pages.

The workflow presented in this use case is general and reusable. The query chosen for monitoring was arbitrary and can be trivially changed. This type of workflow could also be used for monitoring sentiment on other subjects, such as monitoring the Twitter sentiment of political candidates during an election, or monitoring the sentiment of financial tweets with stock symbols as queries.

VI. CONCLUSION AND FURTHER WORK

This paper presents a data mining platform for constructing and executing scientific workflows, and the adaptation of this platform to suit the needs of real-time data processing. It implements the visual programming paradigm, which is used for presenting complex procedures as a sequence of simple steps, which renders the platform usable for non-experts.

We have demonstrated the simplicity and usability of the platform on a Twitter sentiment analysis use case which collects tweets for a given query, filters the tweets based on language and classifies the sentiment of the tweet as positive or negative. The platform produces results in forms of a word cloud and a line chart displaying the positive sentiment, the negative sentiment, and the difference between the positive and negative sentiment. The workflow constructed in this use case is reusable and can be executed for any given query to monitor sentiment on Twitter for arbitrary subjects. While this use case required implementing some new workflow components to work with tweets and the Twitter API, these components are now available and completely reusable in other stream mining workflows.

ClowdFlows currently boasts more than two hundred widgets for batch processing and only fourteen for stream mining. It is our aim to create more specialized widgets for stream mining so that they can be used by practitioners without the need to develop their own components.

In future work, we will also add mechanisms to ensure fault-tolerance and guarantee the processing of all the data from the incoming streams. We plan to release the sources of the platform under an open source licence, and develop scripts that will support single-click deployment of ClowdFlows to cloud services such as Amazon Elastic Compute Cloud (EC2).

ACKNOWLEDGMENT

This work was supported by the FP7 European Commission projects “Machine understanding for interactive storytelling” (MUSE, grant agreement no: 296703) and “Large scale information extraction and integration infrastructure for supporting financial decision making” (FIRST, grant agreement 257928).
The authors would like to thank Matjaž Juršič for his help in developing ClowdFlows, and Jasmina Smailović for developing the SVM sentiment classifier.

REFERENCES


