Information Processing and Management 51 (2015) 187–203

Contents lists available at ScienceDirect

Information Processing and Management

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

Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the ClowdFlows platform

Janez Kranjc [a,b,*], Jasmina Smailović [a,b], Vid Podpečan [a,c], Miha Grčar [a], Martin Žnidaršič [a], Nada Lavrač [a,b,d]

[a] Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia
[c] University of Ljubljana, Faculty of Mathematics and Physics, Jadranska 19, 1000 Ljubljana, Slovenia
[d] University of Nova Gorica, Vipavska 13, 5000 Nova Gorica, Slovenia

ARTICLE INFO

Article history:
Received 30 October 2013
Received in revised form 27 March 2014
Accepted 7 April 2014
Available online 1 May 2014

Keywords:
Active learning
Stream mining
Sentiment analysis
Stream-based active learning
Workflows
Data mining platform

ABSTRACT

Sentiment analysis from data streams is aimed at detecting authors' attitude, emotions and opinions from texts in real-time. To reduce the labeling effort needed in the data collection phase, active learning is often applied in streaming scenarios, where a learning algorithm is allowed to select new examples to be manually labeled in order to improve the learner's performance. Even though there are many on-line platforms which perform sentiment analysis, there is no publicly available interactive on-line platform for dynamic adaptive sentiment analysis, which would be able to handle changes in data streams and adapt its behavior over time. This paper describes ClowdFlows, a cloud-based scientific workflow platform, and its extensions enabling the analysis of data streams and active learning. Moreover, by utilizing the data and workflow sharing in ClowdFlows, the labeling of examples can be distributed through crowdsourcing. The advanced features of ClowdFlows are demonstrated on a sentiment analysis use case, using active learning with a linear Support Vector Machine for learning sentiment classification models to be applied to microblogging data streams.

1. Introduction

This paper addresses a data mining scenario at the intersection of active learning, sentiment analysis, stream mining and service-oriented knowledge discovery architectures effectively solved by on-line workflow implementation of the developed active learning methodology for sentiment analysis from streams of Twitter data.

Active learning is a well-studied research area (Sculley, 2007; Settles, 2011; Settles & Craven, 2008), addressing data mining scenarios where a learning algorithm can periodically select new examples to be labeled by a human annotator and add them to the training dataset to improve the learner's performance on new data. Its aim is to maximize the performance of the algorithm and minimize the human labeling effort. *Sentiment analysis* (Liu, 2012; Pang & Lee, 2008; Turney, 2002) is concerned with the detection of the author's attitude, emotion or opinion about a given topic expressed

* Corresponding author at: Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia. Tel.: +386 1 477 3655.
E-mail address: janez.kranjc@ijs.si (J. Kranjc).

http://dx.doi.org/10.1016/j.ipm.2014.04.001
0306-4573/

in the text. The task of sentiment analysis is especially challenging in the context of analyzing user generated content from the Internet (Petz et al., 2012, 2013). *Stream mining* (Gama, Rodrigues, Spinosa, & de Carvalho, 2010) is an online learning paradigm, aiming to incorporate the information from the evolving data stream into the model, without having to re-learn the model from scratch; while batch learning is a finite process that starts with a data collection phase and ends with a stationary model, the online learning process starts with the arrival of some training instances and lasts as long as there is new data available for learning. As such, it is a dynamic process that has to encapsulate the collection of data, the learning and the validation phase in a single continuous cycle.

This paper introduces a cloud-based scientific workflow platform, which is able to perform on-line dynamic adaptive sentiment analysis of microblogging posts. Even though there are many on-line platforms which apply sentiment analysis on microblogging texts, there is still no such pltaform that could be used for on-line dynamic adaptive sentiment analysis and would thus be able to handle changes in data streams and adapt its components over time. In order to provide continuous updating of the sentiment classifier with time we used an active learning approach. In this paper, we address this issue by presenting an approach to interactive stream-based sentiment analysis of microblogging messages in a cloud-based scientific workflow platform ClowdFlows.[1] With the aim to minimize the effort required to apply labels to tweets, this browser-based platform provides an easy way to share the results and a Web interface for labeling tweets.

ClowdFlows is a new open-sourced data mining platform designed as a cloud-based Web application in order to overcome several deficiencies of similar data mining platforms, providing a handful of novel features that benefit the data mining community. ClowdFlows was first developed as a data mining tool for processing static data (Kranjc, Podpecian, & Lavraci, 2012a, 2012b), 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. ClowdFlows also benefits from a service-oriented architecture which allows users to utilize arbitrary Web services as workflow components. In this paper we present the adaptation of the ClowdFlows platform, enabling it to work on real time data streams. As a result, workflows in ClowdFlows are no longer limited to static data on the server but can connect to multiple data sources and can process the data continuously. One such data source is the Twitter API which provides a potentionally infinite stream of tweets which are the subject of sentiment analysis in this paper.

The paper is structured as follows. Section 2 presents the related work. Comparable data mining and stream mining platforms are presented and their differences and similarities with ClowdFlows are discussed. Related work concerning active learning in data streams is also presented. Section 3 presents the technical background and implementation details of the ClowdFlows platform. The architecture of the system is presented along with specific methods that allow stream mining in a workflow environment. The proposed sentiment analysis and active learning methods are presented in Section 4. The implementation details and the workflow enabling active learning for sentiment analysis are presented in Section 5. In Section 6 we conclude the paper by presenting the directions for further work.

2. Related work

This section presents an overview of data mining platforms and their key features: visual programming and execution of scientific workflows, diversity of workflow components, service-oriented architectures, remote workflow execution, big data processing, stream mining, and data sharing. The overview is followed presenting current research in the field of active learning on data streams.

2.1. Data mining platforms

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 to easily compare different options. All major data mining platforms, such as Weka (Witten, Frank, & Hall, 2011), RapidMiner (Mierswa, Wurst, Klinkenberg, Scholz, & Euler, 2006), KNIME (Berthold et al., 2007) and Orange (Demšar, Zupan, Leban, & Curk, 2004) support workflow construction. The most important common feature is the implementation of a *workflow canvas* where complex workflows can be constructed using simple drag, drop and connect operations on the available components. The range of available components typically includes database connectivity, data loading from files and preprocessing, data and pattern mining algorithms, algorithm performance evaluation, and interactive and non-interactive visualizations.

Even though such data mining software solutions are reasonably user-friendly and offer a wide range of components, some of their deficiencies severely limit their utility. Firstly, all available workflow components provided by any of these platforms are specific and can be used only in the given platform. Secondly, the described platforms are implemented as standalone applications and have specific hardware and software dependencies. 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.

[1] http://clowdflows.org.

As a benefit of 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 grid deployment and P2P computing). Environments such as Weka4WS (Talia, Trunfio, & Verta, 2005), Orange4WS (Podpečan, Zemenova, & Lavrač, 2012), Web Extension for RapidMiner, Triana (Taylor, Shields, Wang, & Harrison, 2007), Taverna (Hull et al., 2006) and Kepler (Altintas et al., 2004) 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 to domains like systems biology, chemistry, medical imaging, ecology and geology. Lastly, all mentioned platforms are still based on technologies that do not benefit from modern Web technologies which enable truly independent software solutions. On the other hand, Web-based workflow construction environments exist, which are however too general and not coupled to any data mining library. For example, Oryx Editor (Decker, Overdick, & Weske, 2008) can be used for modeling business processes and workflows while the Galaxy (Blankenberg et al., 2001, chap. 19) genome analysis tool (implemented as a Web application) is limited exclusively to the workflow components provided by the project itself.

Remote workflow execution (on different machines than the one used for workflow construction) is employed by KNIME Cluster execution 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.

In support of the ever increasing amount of data several truly distributed software platforms have emerged. 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,[2] an open-source implementation of the MapReduce programming model (Dean & Ghemawat, 2008) and a distributed file system called Hadoop Distributed Filesystem (HDFS). It is used in many real life environments and several modifications and extensions exist, also for online (stream) processing (Condie et al., 2010) (parallelization of a variety of learning algoriths using an adaptation of MapReduce is discussed by Chu et al. (2006)). Apache Hadoop is also the base framework of Apache Mahout,[3] a machine learning library for large data sets, which currently supports recommendation mining, clustering, classification and frequent itemset mining. Radoop,[4] a commercial big data analytics solution, is based on RapidMiner and Mahout, and uses RapidMiner's data flow interface.

For data stream processing, two of the most known platforms were released by Yahoo! (the S4 platform[5]) and Twitter (Storm[6]). SAMOA (Morales, 2013) is an example of a new generation platform which is targeted at processing big data streams. In contrast with 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 the most common tasks such as classification and clustering. However, the platform is under development, no software has been released yet and it is not known whether the platform will support visual programming with workflows. MOA (Massive On-line Analysis) is a non-distributed framework for mining data streams (Bifet, Holmes, Kirkby, & Pfahringer, 2010). It is related to the WEKA project and bi-directional interaction of the two is possible. MOA itself does not support visual programming of workflows but the ADAMS project (Reutemann & Vanschoren, 2012) 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 OpenML Experiment Database (Vanschoren & Blockeel, 2009), 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.

2.2. Active learning for data streams

There exist three different scenarios for active learning: (i) membership query synthesis, (ii) pool-based sampling, and (iii) stream-based selective sampling (Settles, 2010). In the membership query synthesis scenario, the learning algorithm can select examples for labeling from the input space or it can produce new examples itself. In the pool-based scenario, the learner has access to a collection of previously seen examples and may request labeling for any of them. In this study, we are interested in the third scenario: the stream-based active learning approach. More specific, we are interested in the active learning on stream data for sentiment analysis of Twitter posts. In this scenario, examples are constantly arriving from a data stream and the learning algorithm has to decide in real time whether to select an arriving example for labeling or not. Therefore, the approach which would handle this scenario has to:

- have constant access to a source of data,
- have the ability to quickly and in real time process each incoming instance and decide whether to request a label for it,
- periodically update the model and apply it to new instances.

[2] http://hadoop.apache.org/.
[3] http://mahout.apache.org/.
[4] http://www.radoop.eu.
[5] http://incubator.apache.org/s4/.
[6] http://storm-project.net/.

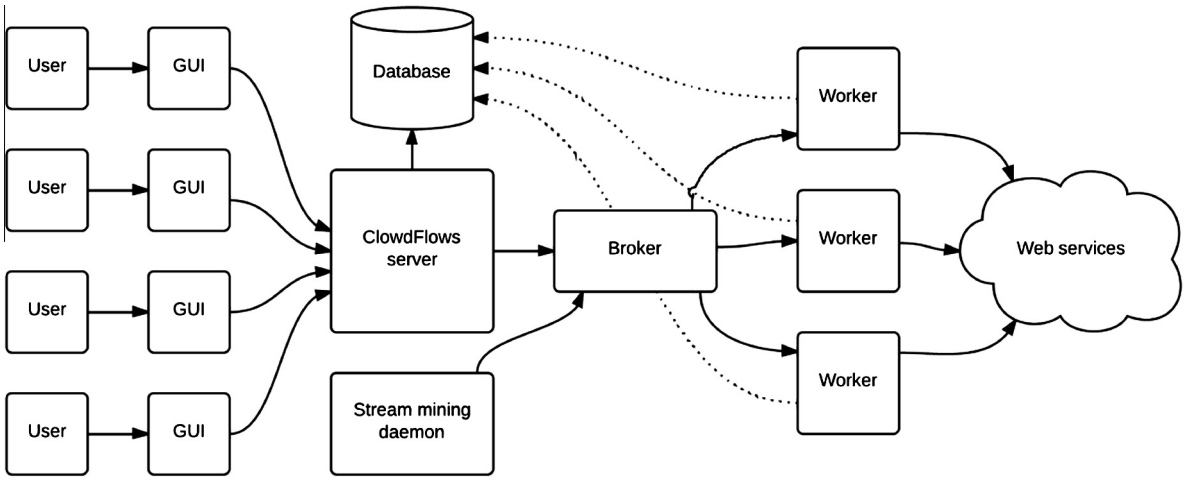

Fig. 1. An overview of the ClowdFlows platform design.

In the stream-based active learning setting, there exist several approaches to deciding whether or not to request hand labels for examples which come from a data stream. One of the simplest strategies is to use some informativeness measure and request labeling for the examples which are the most informative. For instance, the examples for which the learner has the highest uncertainty can be considered the most informative and be selected for labeling. Zhu, Zhang, Lin, and Shi (2007) used uncertainty sampling to label examples within a batch of data from the data stream. Žliobaitė, Bifet, Pfahringer, and Holmes (2011) propose strategies that are based on uncertainty, dynamic allocation of labeling efforts over time and randomization of the search space. Our active learning approach also employs randomization of the search space, but in contrast to the work of Žliobaitė et al. (2011), we organize the examples from the data stream into batches. The decision which examples are best for labeling can be made by a single evolving classifier (Žliobaitė et al., 2011) or by a classifier ensemble (Wang, Zhang, & Guo, 2012; Zhu et al., 2007, Zhu, Zhang, Lin, & Shi, 2010). In our study, we use a single evolving sentiment classifier for Twitter posts.

Our preliminary work on active learning on stream data for sentiment analysis of tweets is presented in (Saveski & Grčar, 2011). The closely related contribution was made in (Settles, 2011), where the author demonstrated the application of DUALIST, an active learning annotation tool, to Twitter sentiment analysis. The author intended to show the generality of the annotation tool, since it is not adjusted specifically to tweets. On the other hand, our approach is particularly adjusted to Twitter data. Regarding the on-line platform which would handle active learning on stream data for sentiment analysis of Twitter posts, to best of our knowledge, we are the first addressing this issue.

3. The ClowdFlows platform

In this section the ClowdFlows platform is presented. The enabling technologies are presented briefly and displayed in the architecture of the system. To validate the design of the platform we present a stress test with many simultaneous users executing their workflows. The graphical user interface and the workflow model are presented. Finally the real-time analysis features of ClowdFlows are described.

3.1. Platform design

As a new generation data mining platform, ClowdFlows (Kranjc et al., 2012a, Kranjc, Podpečan, & Lavrač, 2012b) is designed and implemented using modern technologies and computing paradigms. It is essentially a cloud-based Web application that can be accessed and controlled from anywhere while the processing is performed in a cloud of computing nodes. To achieve the goal of developing a platform that can be accessed and controlled from anywhere and executed on a cloud, we have designed it as a cloud-based Web application. As such it can be, based on the technologies used, logically separated on two sides – the client side, and the server side. The architecture of the platform accessed by multiple users is shown in Fig. 1. A similar architecture figure was previously published in (Kranjc et al., 2012a), with some major differences. In contrast to the previously published architecture, the platform now features a relational database for storing workflows, a broker for delegating tasks to worker nodes and a stream mining daemon for processing data streams.

The client side of the platform consists of operations that involve user interaction. The user interacts with the platform primarily through the graphical user interface in a contemporary Web browser. We have implemented the graphical user

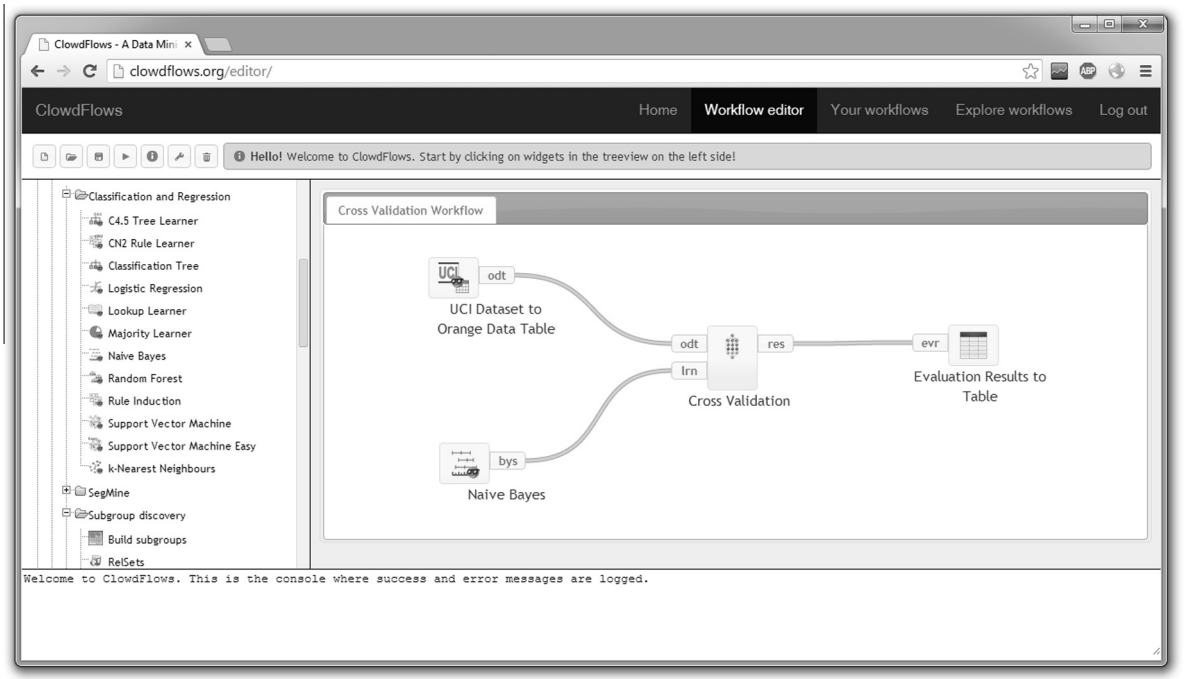

Fig. 2. A screenshot of the ClowdFlows graphical user interface loaded in the Google Chrome Web browser.

interface in HTML and JavaScript, with an extensive use of the jQuery library.[7] The jQuery library was designed to simplify client-side scripting, and is the most popular JavaScript library in use today.[8]

The server side is written in Python and uses the Django Web framework.[9] Django is a high level Python Web framework that encourages rapid development and provides an object-relational mapper and a powerful template system. The object-relational mapper 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. MySQL is used in the public installation of ClowdFlows.

In order to allow consumption of Web services and importing them as workflow components, the PySimpleSoap library[10] 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 services as workflow components, and exposing entire workflows as WSDL Web services.

ClowdFlows may also be installed on multiple computers, which is enabled by using the RabbitMQ[11] messaging server and a Django implementation of Celery,[12] a distributed task queue, which allows passing asynchronous tasks between servers. With this tools it is possible to install ClowdFlows on worker nodes which execute workflows. To demonstrate the scalability of the platform with these tools we have performed a stress test which we describe in Section 3.2.

ClowdFlows is publically available for use at http://clowdflows.org. The source code is open sourced under the General Public Licence and can be downloaded at http://github.com/janezkranjc/clowdflows. Detailed installation instructions are provided with the source code.

3.2. Scalability of the platform

In order to test the scalability of the ClowdFlows platform and validate the design decisions described in Section 3.1 enabling big data analytics for data streams we have performed a stress test in which we simulated several users executing their workflows simultaneously and measured the average execution time.

The test was conducted on a simplified workflow that performs 10-fold cross validation with the Naive Bayes algorithm. The workflow is shown in Fig. 2. In order to simulate concurrent users we have implemented a simulation of a user that

[7] http://jquery.com.
[8] http://w3techs.com/technologies/overview/javascript_library/all.
[9] https://www.djangoproject.com.
[10] https://code.google.com/p/pysimplesoap/.
[11] http://www.rabbitmq.com/.
[12] http://celeryproject.org/.

Table 1
Average response time for the execution of the workflow based on different numbers of concurrent users and different setups of worker nodes. The cells display the average execution time in seconds (plus the standard deviation) and the number of workflow executions done by all the worker nodes in the time of the test in parantheses.

Users	Worker nodes			
	1×8	2×8	3×8	$3 \times 8 + 1 \times 16$
1	3.466 ± 0.603 (18)	3.252 ± 0.010 (19)	3.255 ± 0.011 (19)	3.248 ± 0.011 (19)
10	5.665 ± 2.927 (109)	3.913 ± 1.436 (157)	4.012 ± 1.399 (154)	3.476 ± 0.779 (179)
20	9.369 ± 5.449 (130)	5.530 ± 2.786 (222)	5.090 ± 4.189 (236)	4.268 ± 1.742 (290)
50	17.176 ± 8.105 (182)	9.856 ± 6.615 (303)	7.433 ± 4.207 (407)	6.154 ± 3.799 (495)
100	27.534 ± 13.071 (238)	18.088 ± 9.132 (351)	12.745 ± 6.732 (499)	9.882 ± 5.990 (617)

executes her workflow continuously for 60 s without a pause. This settings is also equivalent to executing a streaming process for 1 min. After 60 s have passed the user waits until the final workflow execution results are returned. We tested the platform against different sets of concurrent users: 1 user, 10 users, 20 users, 50 users, and 100 users for different setups of the worker nodes.

A worker node is a headless installation of the ClowdFlows platform that executes workflows. We tested the platform with a single worker node, two worker nodes, and three worker nodes. Each of these worker nodes was installed on equivalent computers with 8 cores. The workers were setup to work on 8 concurrent threads. For the final test we have setup an additional worker on a computer with 16 cores to run 16 threads. We have measured the execution times for each workflow and calculated the average execution time from the beginning of the request until the result was received. The results are shown in Table 1.

The results show that a single user continuously executing her cross validation workflow will be able to execute it 18 or 19 times on any setup if she is the only user executing workflows. In order for ten concurrent users to execute their workflows at a comparable speed at least two worker nodes are needed. The most efficient current setup is three workers with 8 threads each and one worker with 16 threads which still allows a hundred users to issue workflow execution requests to the platform at a reasonable response time.

The platform has succesfully passed the stress test. The results show that the ClowdFlows platform can serve many concurrent users that continuously execute workflows. The average execution times can be controlled by adding or removing worker nodes. The worker nodes can be added and removed during runtime, which means that heavy loads can be resolved simply by adding more computing power to the ClowdFlows worker cluster. As adding worker nodes at times with lower loads does not improve the average processing time, we would like to implement a mechanism for automatically spawning and removing worker nodes on services such as the Amazon Elastic Compute Cloud in future work.

3.3. The workflow model

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

3.4. The graphical user interface

The graphical user interface used for constructing workflows follows a visual programming paradigm which simplifies the representation of complex procedures into a spatial arrangement of building blocks. The building blocks (workflow components) in ClowdFlows are referred to as widgets. The graphical user interface implements an easy to use way to arrange widgets on a canvas to form a graphical representation of a procedure. The ClowdFlows graphical user interface rendered in a Web browser is shown in Fig. 2.

The graphical user interface of the ClowdFlows system consists of a workflow canvas and a widget repository. The widget repository is a set of widgets ordered in a hierarchy of categories. Upon clicking on a widget in the repository, that widget appears on the canvas. The workflow canvas implements moving, connecting, issuing commands to execute and delete widgets. Widgets can be arbitrarily arranged on the canvas by dragging and dropping. Connections between widgets can be added by selecting an output of a widget and an input of another widget.

Information on each operation the user performs on the workflow canvas is sent to the server using an asynchronous HTTP POST request. The operation is validated on the server and a success or error message with additional information is passed to the user interface (the client's browser) formatted in JavaScript Object Notation (JSON) or HTML.

On the top of the graphical user interface is a toolbar where entire workflows can be saved, deleted, and executed.

3.5. The widget repository

Widgets in ClowdFlows are separated into four groups based on their purpose: regular widgets, visualization widgets, interactive widgets and workflow control widgets.

Regular widgets perform specific tasks that transform the data from the inputs and the parameters to data on the outputs, and provide success or error messages to the system. The task of a widget is written as a Python function that takes a Python dictionary of inputs and parameters as its arguments and returns a dictionary of outputs. The function is called each time the widget is executed. Widgets that implement complex procedures can also implement a progress bar, that displays progress to the user in real time.

Visualization widgets are extended versions of regular widgets as they also provide the ability to render an HTML template with JavaScript to the client's browser. These are useful for data visualizations and presentation of more detailed feedback to the user. Visualization widgets are regular widgets with the addition of a second Python function which controls the rendering of the template. This function is only invoked when the workflow is executed from the user interface.

An interactive widget is a widget that requires data before execution in order to prompt the user for the correct parameters. These widgets are extensions of regular widgets as they perform three functions. The data preparation function executes first and takes the inputs and parameters as the arguments. The second function is a rendering function where a modal window is prepared by using an HTML template which prompts the user to manipulate the data. The final function's arguments are the user's input and the inputs and parameters of the widget. A widget can also be a combination of an interactive and a visualization widget, where it executes a fourth rendering function to display the results.

Three special widgets provide additional workflow controls. These are the *Sub-workflow*, *Input*, and *Output* widget. Whenever a *Sub-workflow* widget is added to a workflow, an empty workflow is created that will be executed when the sub-workflow widget is executed. The *Sub-workflow* widget has no inputs and outputs by default, so they have to be added to the workflow by the user using the *Input* and the *Output* widget. Whenever an *Input* or *Output* widget is put on a workflow that is a sub-workflow of another workflow, an actual input or output is added to the widget representing the sub-workflow. Workflows can be indefinitely nested this way.

Two variations of the input and output widget provide ways to loop through sub-workflows. The input and output widgets can be replaced by the *For Input* and *For Output* widgets. Whenever a workflow contains these two widgets, the workflow execution engine will attempt to break down the object on the input and execute the workflow once for each piece of data that is on the input. With these controls a workflow can be executed on a list or array of data.

3.6. The workflow execution engine

The job of the workflow execution engine is to execute all executable widgets in the workflow in the correct order. The engine is implemented twice, both in Python and JavaScript due to performance issues when the user wishes to see the order of the executed widgets in real time.

The two implementations of the workflow execution engine are similar with two differences. The JavaScript engine is enabled by default due to the requests for executing separate widgets being asynchronous HTTP requests. Each request is handled by the server separately and executes a single widget, saves the changed and returns the results to the client where the execution continues. The server side Python implementation only receives one HTTP request for the entire workflow and multiprocessing had to be implemented manually. For performance issues, sub-workflows and loops are executed by the Python implementation, while top-level workflows executed from the user interface are processed by the JavaScript implementation. The JavaScript implementation shows the results of the execution of each widget in real time, while the user can only see the results of the Python implemented workflow execution after it has finished in full.

When a workflow is running, the execution engine perpetually checks for widgets that are executable and executes them. Executable widgets are widgets which either have no predecessors, or their predecessors have already been successfully executed. Whenever two or more widgets are executable at the same time they are asynchronously executed in parallel, since they are independent. The implemented 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.

3.7. Public workflows

Since workflows in ClowdFlows are processed and stored on remote servers they can be accessed from anywhere with an internet connection. 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 specific URL for each workflow that has been saved as public. Users can then simply share their workflows by publishing the URL. Whenever a public workflow is accessed by a 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

repeatability of experiments. Each such copied public workflow can also be edited, augmented or used as a template to create a new workflow, which can be made public as well.

3.8. Real-time data analysis in ClowdFlows

In comparison with the early implementations of the ClowdFlows platform described in (Kranjc et al., 2012a, 2012b) the novelty of this work is the ability of ClowdFlows to process real-time data streams. Its workflow engine has been augmented with continuous parallel execution and the halting mechanism and several specialized widgets for stream data processing were developed. In the following we describe the new data stream processing capabilities of the ClowdFlows platform.

3.8.1. 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 the results from special *stream visualization* widgets.

3.8.2. Specialized workflow widgets for real-time processing

Widgets in stream mining workflows have, in contrast to widgets in regular workflows, 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* 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 had 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.

Several other popular stream mining approaches (Ikonomovska, Loskovska, & Gjorgjevik, 2007) 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 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 the results can be shared without having to share the actual workflows.

4. Active learning for sentiment analysis

In this section we first describe the dataset we use for the default tweet sentiment classifier, preprocessing techniques and the algorithm for sentiment analysis. The approach to tweet preprocessing and classifier training is implemented using the LATINO[13] software library of text processing and data mining algorithms. The section continues with a description of the active learning algorithm and the strategy used to select data instances for labeling.

4.1. The data used for the default sentiment classifier

The default tweet sentiment classifier is trained on a collection of 1,600,000 (800,000 positive and 800,000 negative) tweets collected and prepared by Stanford University (Go, Bhayani, & Huang, 2009), where the tweets were labeled based on positive and negative emoticons in them. Therefore, the emoticons approximate the actual positive and negative sentiment labels. This approach was introduced by Read (Read, 2005). If a tweet contains ":)", ":-)", ":)", ":D" or "=)" emoticon it was labeled as positive, and if it contains ":(", ":-(" or ": (" emoticon it was labeled as negative. In the training data, the tweets containing both positive and negative emoticons, retweets and duplicate tweets were removed (Go et al., 2009). The emoticons, which approximate sentiment labels, were also already removed from the tweets in order not to put too much weight on them in the training phase, and therefore the classifier learns from the other features of tweets. The tweets in this collection do not belong to any particular domain.

4.2. Data preprocessing

Preprocessing of data is an important step when using supervised machine learning techniques. On the Twitter data, we apply both standard and Twitter-specific text preprocessing to better define the feature space. The specific text preprocessing is especially important for Twitter messages, since user generated content on the Internet often contains slang (Petz et al., 2012) and messages from social media are considered noisy, containing many grammatical and spelling mistakes (Petz et al., 2013). Therefore, with our Twitter-preprocessing, we try to overcome these problems and improve the quality of features.

As a part of the Twitter preprocessing step (Agarwal, Xie, Vovsha, Rambow, & Passonneau, 2011; Go et al., 2009; Smailović, Grčar, Lavrač, & Žnidaršič, 2013; Smailović, Grčar, & Žnidaršič, 2012) we replace mentioning of other Twitter users in a tweet of the form @TwitterUser by a single token named *USERNAME* and writing different Web links by a single token named *URL*. Moreover, letters which repeat for more than two times are replaced by one occurrence of such letter; for example, the word *looooooooove* is transformed to *love*. We replace negation words (*not*, *isn't*, *aren't*, *wasn't*, *weren't*, *hasn't*, *haven't*, *hadn't*, *doesn't*, *don't*, *didn't*) with a single token named *NEGATION*. Finally, exclamation marks are replaced by a single token *EXCLAMATION* and question marks by a single token *QUESTION*.

Besides the Twitter-specific text preprocessing, we also apply standard preprocessing techniques (Feldman & Sanger, 2007) in order to better define and reduce the feature space. These involve text tokenization (text splitting into individual words/terms), stopwords removal (removing words which do not contain relevant information, e.g., *a*, *an*, *the*, *and*, *but*, *if*, *or*, etc.), stemming (converting words into their base or root form) and N-gram construction (concatenating 1 to N stemmed words appearing consecutively in a tweet). The resulting terms are used as features in the construction of feature vectors representing the tweets, where the feature vector construction is based on term frequency feature weighting scheme. We do not apply a part of speech (POS) tagger, since it was indicated by Go et al. (2009) and Pang and Lee (2002) that POS tags are not useful when using SVMs for sentiment analysis. Also, Kouloumpis, Wilson, and Moore (2011) showed that POS features may not be useful for sentiment analysis in the microblogging domain.

4.3. The algorithm used for sentiment classification

Sentiment analysis methods (Liu, 2012; Pang & Lee, 2008; Turney, 2002) aim at detecting the authors attitude, emotions or opinion about a given topic expressed in text. There are three generally known approaches to sentiment analysis (Pang & Lee, 2008; Thelwall, Buckley, & Paltoglou, 2011): (i) machine learning, (ii) lexicon-based methods and (iii) linguistic analysis.

We use a machine learning approach, applying the linear Support Vector Machine (SVM) (Cortes & Vapnik, 1995; Vapnik, 1995, 1998), which is a typical algorithm used in document classification. The SVM training algorithm represents the labeled training examples as points in the space and separates them with a hyperplane. A hyperplane is placed in such a way that the examples of the separate classes are divided from each other as much as possible. New examples are then mapped into the same space and classified based on the side of the hyperplane they are. For training the tweet sentiment classifier, we use the SVM^{perf} (Joachims, 2005, 2006; Joachims & Yu, 2009) implementation of the SVM algorithm. In order to test its classification accuracy, we trained the SVM classiffier on the collection of 1,600,000 smiley labeled tweets (Go et al., 2009) and tested it on 177 negative and 182 positive manually labeled tweets, prepared and labeled by Stanford University

[13] LATINO (Link Analysis and Text Mining Toolbox) is open-source—mostly under the LGPL license—and is available at http://latino.sourceforge.net/.

Table 2
Average accuracy, precision and recall in the setting without active learning and with active learning while experimenting with different proportions of random tweets (#rnd) and tweets which are closest to the SVM hyperplane (#hyp) from every batch, containing 1000 tweets, for hand labeling.

Setting	#rnd	#hyp	Accuracy	Precision positive class	Recall positive class	Precision negative class	Recall negative class
No active learning	0	0	0.349	0.463	0.569	0.223	0.643
Active learning	0	100	0.406	0.456	0.829	0.272	0.351
Active learning	25	75	0.413	0.454	0.858	0.275	0.296
Active learning	50	50	0.410	0.459	0.837	0.256	0.335
Active learning	75	25	0.418	0.451	0.881	0.279	0.265
Active learning	100	0	0.416	0.448	0.886	0.288	0.251

(Go et al., 2009). We applied both standard and Twitter specific preprocessing. In this experiment we achieved the accuracy of 83.01% (which is a comparable result with (Go et al., 2009)).

The reason for using a machine learning approach and not lexicon-based or linguistic methods is the following. In the context of active learning for sentiment analysis on data streams, the linguistic methods pose several challenges, as they tend to be too computationally demanding for the use in a streaming near real time setting. Also, there is the lack of readily available tools for parsing tweets. On the other hand, lexicon-based methods are faster, but they usually rely on explicit notion of sentiment and dismiss the terminology that bears sentiment more implicitly. For example, the word 'Greece' bears negative sentiment in the light of the financial crisis, but in general it is neutrally or even positively connoted word.

Nevertheless, in order to compare lexicon and machine learning methods, we have tested a lexicon method classification accuracy on the same collection of 177 negative and 182 positive manually labeled tweets (Go et al., 2009), as for the machine learning approach. In the lexicon-based method, we used an opinion lexicon containing 2006 positive and 4783 negative words[14] (Hu & Liu, 2004; Liu, Hu, & Cheng, 2005). The lexicon is adjusted to social media content, as it also contains many misspelled words which are frequently used in social media language. We applied Twitter specific preprocessing on the test tweets and calculated positive and negative score for each tweet, based on the occurrences of positive and negative lexicon words in them. For example, if a tweet contains a word 'love' from the positive lexicon list, the positive score will increase by one. The score will not increase if the currently observed lexicon word contains or it is contained in some of the previously seen lexicon words for that specific class in the observed tweet. For example, a tweet could contain a word 'nicely'. On the other hand, the positive word lexicon list contains both 'nice' and 'nicely' words. The algorithm will detect that the word 'nice' is presented in the tweet and it will increase the positive score by one. Next, it will check the presence of the word 'nicely' and it will find out that the tweet contains this word, but this word contains a word ('nice'), which already increased the positive score for this tweet, and therefore it will not increase the positive score. If the resulting positive score for a tweet is the same or higher than the negative score, the tweet is labeled as positive. If it is lower, it is labeled as negative. The tweets with equal positive and negative score are labeled as positive, since the positive lexicon list contains less words. In this experiment we achieved the accuracy of 76.04% on the test set.

Since the accuracy on the test set obtained with the machine learning approach was higher than the accuracy obtained with the lexicon-based approach, we decided to focus on the machine learning approach in our study.

4.4. Active learning

In active learning, the learning algorithm periodically asks an oracle (e.g., a human annotator) to manually label the examples which he finds most suitable for labeling. Using this approach and an appropriate query strategy, the number of examples that need to be manually labeled is largely decreased. Typically, the active learning algorithm first learns from an initially labeled collection of examples. Based on the initial model and the characteristics of the newly observed unlabeled examples, the algorithm selects new examples for manual labeling. After the labeling is finished, the model is updated and the process is repeated for the new incoming examples. This procedure is repeated until some threshold (for example, time limit, labeling quota or target performance) is reached or, in the case of data streams, it continues as long as the application is active and new examples are arriving.

In our software, the active learning algorithm first learns from the Stanford smiley labeled data set as an initial labeled data set. According to this initial model, the algorithm classifies new incoming tweets from the data stream as positive or negative. Tweets, which come from the data stream, are split into batches. The algorithm selects most suitable tweets from a first batch for hand-labeling and puts them in a pool of query tweets. The process is repeated for every following batch and every time the pool of query tweets is updated and the tweets in the pool are reordered according to how suitable they are for hand-labeling. When the user decides to conduct manual labeling, she is given a selected number of top tweets from the pool of query tweets for hand-labeling. The user can label a tweet as positive, negative or neutral. After the labeling, labeled tweets are placed in the pool of labeled tweets and removed from the pool of query tweets. Periodically, using the initial and

[14] The opinion lexicon was obtained from http://www.cs.uic.edu/liub/FBS/sentiment-analysis.html.

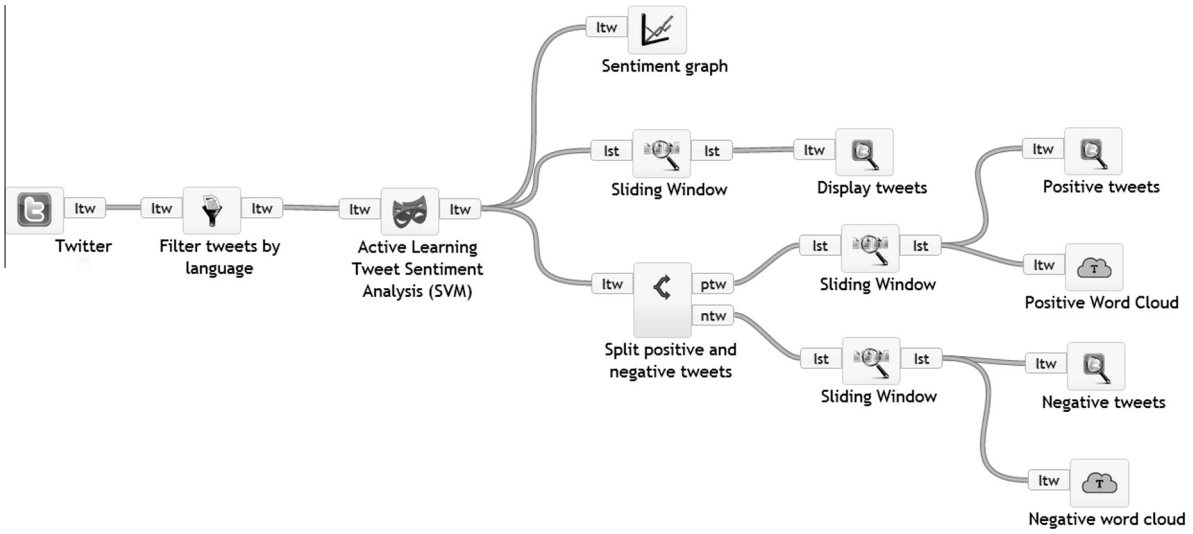

Fig. 3. The Twitter sentiment analysis workflow.

manually positively and negatively labeled tweets from the pool of labeled tweets, the model is retrained. This process is repeated until it is terminated by the user.

The selection of tweets, which are suitable for manual labeling is based on uncertainty strategy and randomization of the search space. The randomization of the search space was also used by Žliobaitė et al. (2011). We experimented with different proportions of random tweets and tweets which are closest to the SVM hyperplane in order to find the best combination of them. Additionally, we performed one experiment in which we did not apply the active learning strategy, i.e., the sentiment classifier was static and did not update over time. In order to automatically conduct these experiments, we hand-labeled a data set of 11,389 financial tweets (4861 positive, 1856 negative and 4672 neutral tweets) discussing the Web search engine provider Baidu,[15] which were collected for a period from March 11 to December 9, 2011. The evaluation method was based on a holdout evaluation approach (Bifet & Kirkby, 2009; Ikonomovska, 2012; Ikonomovska, Gama, & Džeroski, 2011) for data streams where concept drift is assumed. The classifier's performance is tested on a new batch of tweets which come from the data stream. After the testing is finished, the algorithm selects a predefined number of tweets from the same batch and asks an oracle to label them. The newly labeled tweets are added to the training set and used for updating the sentiment model. This procedure is repeated for every new batch of tweets from the data stream. We calculate the accuracy, precision and recall for every batch and at the end of the simulation we report the overall average measures for all the batches. In our off-line evaluation experiments, we select 100 tweets from every batch, which contains 1000 tweets, and then update the model. The results are presented in Table 2. As can be seen from the table, the accuracy of the sentiment classification is higher when the active learning approach is applied. Among the querying strategies, the highest accuracy is obtained by selecting 75 random tweets and 25 tweets which are closest to the SVM hyperplane.

In contrast to the experimental setting described above, the workflow developed for practical use (shown in Fig. 3) by default splits tweets from the data stream into batches which contain 100 tweets, and selects 10 for hand labeling. Following the best strategy from Table 2, the algorithm selects 3 tweets that are closest to the SVM hyperplane and puts them into the pool of query tweets, so that the top most are the ones which are closest to the hyperplane, i.e., the most uncertain ones for the classifier. The other 7 tweets are chosen randomly from the batch and put into a separate pool of random tweets. With time, as new tweets arrive, the pools are updated. Whenever the user decides to label some tweets, she is presented with a set of tweets to label, which contains 3 most uncertain ones from the pool of query tweets and 7 random ones from the pool of random tweets. The hand-labeled tweets are placed in the pool of labeled tweets. Periodically, using the initial and manually labeled tweets from the pool of labeled tweets, the model is retrained.

5. Active learning sentiment analysis workflow implementation in ClowdFlows

In this section we present the implementation of an active learning sentiment analysis use case on Twitter data in the form of an executable workflow. The use case description 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 streaming active learning sentiment analysis process and observe its results.

[15] http://www.baidu.com/.

The aim of the use case is to monitor the Twitter sentiment on a given subject with the possibility to manually label tweets to improve the classification model. 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.

5.1. Rationale

We have decided to implement this stream-mining workflow in the ClowdFlows platform for several reasons.

The execution of the stream-mining workflow is bottlenecked by the rate of incoming Tweets, which is imposed by the Twitter API. Therefore any stream mining platform capable of processing tweets at a higher rate than the API's incoming rate would be as efficient as ClowdFlows for this use case. However, the benefit of using ClowdFlows for this task is its extensible user interface which allows for human–computer interaction during the course of the stream mining process. In this use case the user interface is used during runtime for labeling Tweets. The user interface can also be used to modify the workflow by using the intuitive visual programming paradigm interface. Moreover, the ability to share workflows allows us to publish this workflow on the Web and allow single click deployment of it to the users. The users can also augment, extend or modify the workflow to suit their needs without any coding knowledge just by rearranging the workflow components on the canvas.

5.2. Development of necessary components

To construct the workflow we required a *stream input* widget that can collect tweets based on a query, a *sampling* widget that should 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.

5.2.1. Streaming input, filtering, and visualizations

To consume the incoming stream we implemented a widget that connects to Twitter via the Twitter API.[16] 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.

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, the volume of positive tweets, the volume of negative tweets, and the difference of positive and negative tweets. The visualization was implemented with the HighCharts JavaScript visualization library.[17]

To inspect separate tweets a simple table was implemented where each tweet is colored 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 dynamic and changes with the stream. Looking at the word cloud and seeing popular words appearing and unpopular words disappearing is a novel way to inspect data streams in real-time. The visualization was developed with the D3.js JavaScript library (Bostock, Ogievetsky, & Heer, 2011).

5.2.2. Sentiment classification and active learning

To implement sentiment classification and active learning discussed in Section 4, which was developed in the.NET framework, we exposed it as a Web service that provided several operations:

- classify a set of tweets for a specific workflow,
- return a set of tweets for manual labeling for a specific workflow,
- update a model for a specific workflow.

The service keeps track of multiple workflows and builds a model for workflows separately (in order to better conform the models for specific topics and to avoid malicious labeling affecting the models for legitimate users). Whenever the service is queried a unique identifier of the processing component is also passed along to determine which model to use.

[16] https://dev.twitter.com/.
[17] http://www.highcharts.com/.

The *Classify a set of tweets* operation accepts a set of tweets and an identifier of the processing component at the input. Upon execution it loads the appropriate model and applies it to the tweets. The loading times of the models were reduced to become shorter than the waiting time required to conform to the rate limit of the Twitter API in order to guarantee the processing of all the tweets. The operation returns a set of labels for the tweets.

Return a set of tweets for manual labeling is an operation that accepts the unique identifier of the processing component and returns ten tweets for manual labeling for that specific model. The tweets are then deleted from its pool of query tweets.

The *Update model* operation accepts a set of labeled tweets and a unique identifier of the processing component to update the model. The updating of a model takes several minutes so special care was taken in order to only update models when really necessary.

The functions were implemented into a workflow processing component in the following way: we have developed a streaming workflow component that receives a list of tweets at the input. These tweets are provided by the Twitter API usually in a batch of a hundred or less tweets. The tweets are sent to the *Classify a set of tweets* operation of the Web service. The sentiment labels that are returned from the Web service are appended to the tweets which are sent to the visualization widgets in the workflow. The active learning workflow component also has a special view that functions as an interactive visualization. This view is accessible the same way as other visualizations of the workflow (by special URLs). Whenever this view is accessed the Web service is polled for tweets that require manual labeling. If there are no query tweets in the pool, a friendly message is displayed to the user, prompting her to come back later. If there are query tweets in the pool they are displayed to the user along with a simple form that can be used to manually label the tweets either as positive, negative, or neutral. When the user labels the tweets and clicks the *Submit labels* button, the labeled tweets are saved into the internal memory of the active learning sentiment analysis component. The *Update model* operation is invoked once a day for every streaming workflow that has an active learning widget with new labeled tweets.

5.3. 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 Fig. 3.

Parameters were set after the workflow was constructed. 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 *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 words of 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 2013.

5.4. 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.

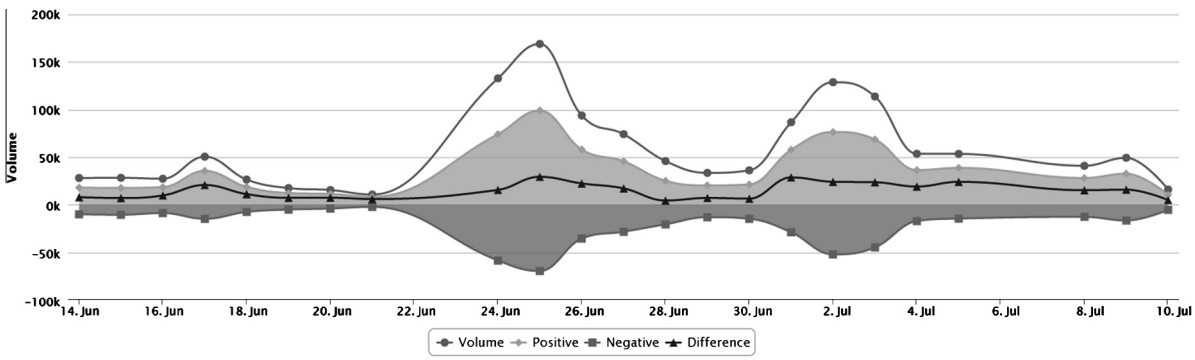

Fig. 4. A line chart of sentiment, volume, and sentiment difference over time.

Fig. 5. A word cloud constructed from tweets with a negative sentiment.

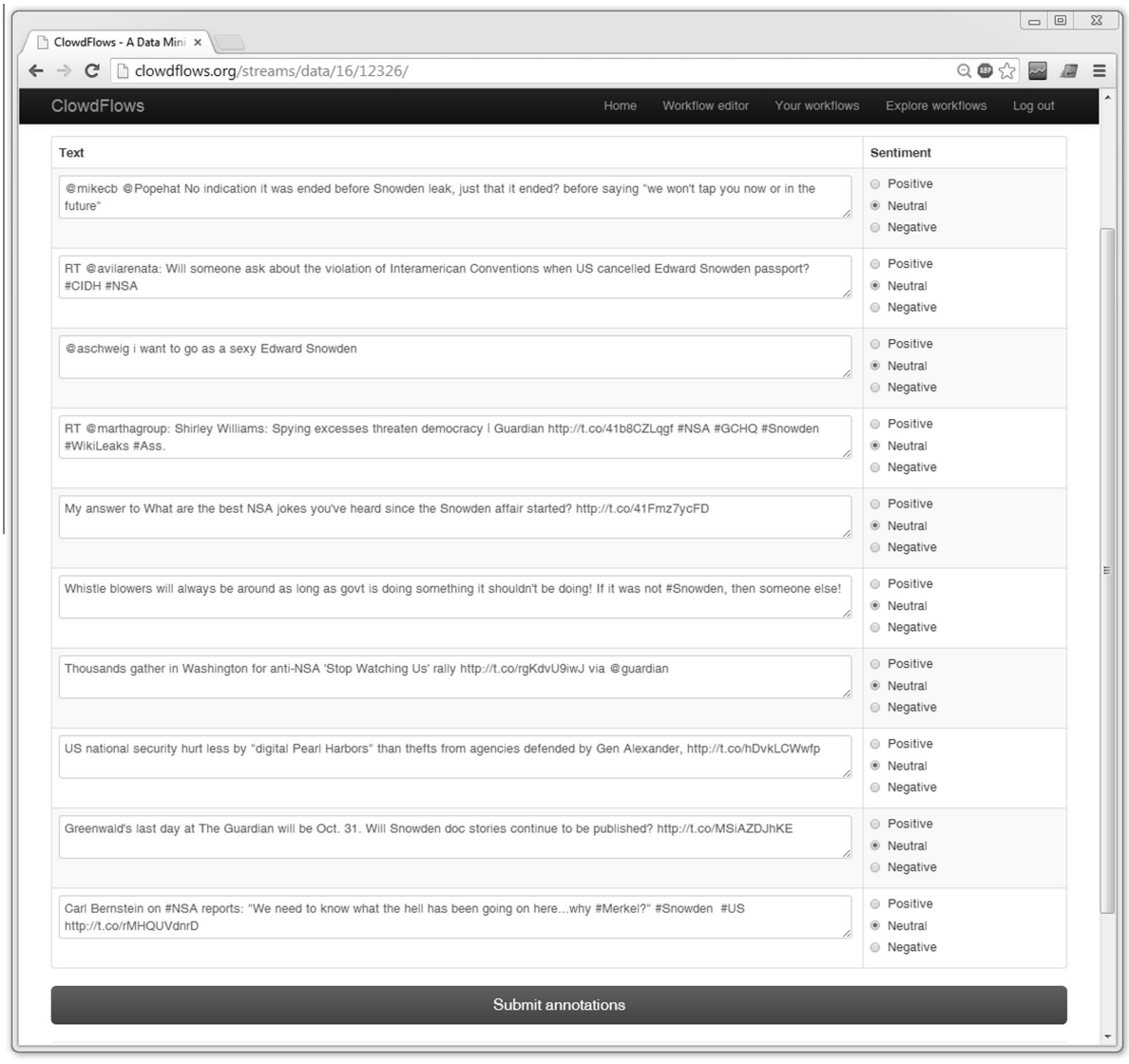

Fig. 6. The tweet labeling user interface.

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 shown in Fig. 4. By looking at this visualization we can see that the sentiment in the tweets mentioning Snowden is generally more positive than negative. We can observe several spikes in the 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.

The word cloud visualization of negative tweets is available at http://clowdflows.org/streams/data/4/9065/ and is shown in Fig. 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.

5.5. Labeling the tweets

Similar to stream visualization widgets the *Active learning sentiment analysis* widget provides a special URL that can be accessed by human annotators. Propagating this link is an easy way to crowdsource labeling of tweets.

The labeling interface for this use case is available at http://clowdflows.org/streams/data/16/12326/ and is shown in Fig. 6. The tweets that require labeling are displayed and users can label them as positive, negative, or neutral. Upon clicking the button *Submit annotations* the labeled tweets are saved into the widget's internal memory. These tweets are accessed and sent to the sentiment analysis Web service once a day, if there are any new labeled tweets on that particular day.

6. Conclusion and further work

We have implemented an active learning scenario for sentiment analysis on Twitter data in a cloud-based data mining platform. In order to do so we adapted the platform to work with data streams by use of two mechanisms: widget memory and the halting mechanism.

We have developed a Web service that utilizes the Support Vector Machine algorithm to build and update sentiment analysis models. The service also applies the models on unlabeled tweets and determines which tweets require manual labeling by the user. We have developed workflow components that utilize this Web service in order to provide an intuitive interface for labeling tweets and setting up new active learning sentiment analysis scenarios from scratch without the need of programming or installing complex software. For each active learning workflow a special Web page is created where tweets can be labeled. By propagating the address of this Web page, crowdsourcing and collaborative knowledge discovery can be utilized to label vast amounts of tweets.

In future work we wish to implement several different strategies for selecting the tweets suitable for labeling and to allow the user to select the most appropriate one. We also wish to allow more control over the generation of the initial models and a richer selection of initially labeled datasets. In the current version of our software, we assume sentiment analysis to be a two class classification problem and classify tweets only as positive or negative, in order to enable simple and efficient calculations in real time. But, tweets can also be neutral, and our current implementation of the software does not allow 3 class classification. In our previous study (Smailović et al., 2013) we introduced a method to classify tweets also as neutral. In future work we plan to adapt and implement this method for inclusion in ClowdFlows.

The source code of the platform is released under an open source licence (GPL) and can be obtained at http://github.com/janezkranjc/clowdflows.

Acknowledgements

This work was supported in part by the European Commission FP7 FET-Open project FOC (Forecasting financial crises, Grant No. 255987), the FP7 European Commission project MUSE (Machine understanding for interactive storytelling, grant agreement no: 296703), 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 by the Slovenian Research Agency through the research program Knowledge Technologies under Grant P2-0103. The research was also supported by Ad Futura Programme of the Slovenian Human Resources and Scholarship Fund.

References

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R. (2011). Sentiment analysis of twitter data. In *Proceedings of the workshop on languages in social media* (pp. 30–38). Association for Computational Linguistics.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., & Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th international conference on scientific and statistical database management* (pp. 423–424). New York: IEEE Computer Society.
Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., et al (2007). 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* (pp. 319–326). Springer.
Bifet, A., & Kirkby, R. (2009). Data stream mining: A practical approach, Citeseer.
Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*, 1601–1604.
Blankenberg, D., Kuster, G. V., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., et al (2001). Galaxy: A web-based genome analysis tool for experimentalists. *Current Protocols in Molecular Biology*.
Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics, 17*, 2301–2309.
Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G. R., Ng, A. Y., et al (2006). Map-reduce for machine learning on multicore. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.), *Proceedings of NIPS* (pp. 281–288). MIT Press.
Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2010). Mapreduce online. In *Proceedings of the 7th USENIX conference on networked systems design and implementation, NSDI'10* (pp. 21–34). Berkeley, CA, USA: USENIX Association.
Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*, 273–297.
Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*, 107–113.
Decker, G., Overdick, H., & Weske, M. (2008). Oryx – An open modeling platform for the bpm community. In M. Dumas, M. Reichert, & M.-C. Shan (Eds.), *Business process management* (pp. 382–385). Springer.
Demšar, J., Zupan, B., Leban, G., & Curk, T. (2004). Orange: From experimental machine learning to interactive data mining. In J.-F. Boulicaut, F. Esposito, F. Giannotti, & D. Pedreschi (Eds.), *Proceedings of PKDD* (pp. 537–539). Springer.
Feldman, R., & Sanger, J. (2007). *The text mining handbook: Advanced approaches in analyzing unstructured data*. Cambridge University Press.
Gama, J., Rodrigues, P. P., Spinosa, E. J., & de Carvalho, A. C. P. L. F. (2010). Knowledge discovery from data streams, Citeseer.
Go, A., Bhayani, R., & Huang, L. (2009). *Twitter sentiment classification using distant supervision*. CS224N Project Report, Stanford (pp. 1–12).
Hu, M., & Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 168–177). ACM.
Hull, D., Wolstencroft, K., Stevens, R., Goble, C. A., Pocock, M. R., Li, P., et al (2006). Taverna: A tool for building and running workflows of services. *Nucleic Acids Research, 34*, 729–732.
Ikonomovska, E. (2012). *Algorithms for learning regression trees and ensembles on evolving data streams*. Ph.D. thesis, Jožef Stefan International Postgraduate School.
Ikonomovska, E., Gama, J., & Džeroski, S. (2011). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery, 23*, 128–168.
Ikonomovska, E., Loskovska, S., & Gjorgjevik, D. (2007). A survey of stream data mining. In *Proceedings of 8th national conference with international participation* (pp. 19–21). ETAI.
Joachims, T. (2005). A support vector method for multivariate performance measures. In *Proceedings of the 22nd international conference on machine learning* (pp. 377–384). ACM.
Joachims, T. (2006). Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 217–226). ACM.
Joachims, T., & Yu, C.-N. J. (2009). Sparse kernel svms via cutting-plane training. *Machine Learning, 76*, 179–193.
Kouloumpis, E., Wilson, T., & Moore, J. (2011). Twitter sentiment analysis: The good the bad and the omg! In *Proceedings of fifth international AAAI conference on weblogs and social media* (pp. 538–541). ICWSM.
Kranjc, J., Podpečan, V., & Lavrač, N. (2012a). Knowledge discovery using a service oriented web application. In *Proceedings of the fourth international conference on information, process, and knowledge management* (pp. 82–87). eKNOW.
Kranjc, J., Podpečan, V., & Lavrač, N. (2012b). Clowdflows: A cloud based scientific workflow platform. In P. A. Flach, T. D. Bie, & N. Cristianini (Eds.), *Proceedings of machine learning and knowledge discovery in databases, ECML/PKDD (2)* (pp. 816–819). Springer.
Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies, 5*, 1–167.
Liu, B., Hu, M., & Cheng, J. (2005). Opinion observer: Analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web* (pp. 342–351). ACM.
Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In L. Ungar, M. Craven, D. Gunopulos, & T. Eliassi-Rad (Eds.), *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 935–940). KDD.
Morales, G. D. F. (2013). SAMOA: A platform for mining big data streams. In L. Carr, A. H. F. Laender, B. F. Lóscio, I. King, M. Fontoura, & D. Vrandecic, et al. (Eds.), *WWW (companion volume)* (pp. 777–778). ACM.
Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval, 2*, 1–135.
Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on empirical methods in natural language processing* (pp. 79–86). Association for Computational Linguistics.
Petz, G., Karpowicz, M., Fürschuß, H., Auinger, A., Stříteský, V., Holzinger, A., et al (2013). In *Proceedings of human–computer interaction and knowledge discovery in complex, unstructured, big data* (pp. 35–46). Springer.
Petz, G., Karpowicz, M., Fürschuß, H., Auinger, A., Winkler, S. M., Schaller, S., et al (2012). On text preprocessing for opinion mining outside of laboratory environments. In *Active media technology* (pp. 618–629). Springer.
Podpečan, V., Zemenova, M., & Lavrač, N. (2012). Orange4WS environment for service-oriented data mining. *The Computer Journal, 55*, 89–98.
Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL student research workshop* (pp. 43–48). Association for Computational Linguistics.
Reutemann, P., & Vanschoren, J. (2012). Scientific workflow management with ADAMS. In P. A. Flach, T. D. Bie, & N. Cristianini (Eds.), *Proceedings of machine learning and knowledge discovery in databases, ECML/PKDD (2)* (pp. 833–837). Springer.
Saveski, M., & Grčar, M. (2011). Web services for stream mining: A stream-based active learning use case, machine learning and knowledge discovery in databases. *ECML PKDD, 2011*, 36.
Sculley, D. (2007). Online active learning methods for fast label-efficient spam filtering. In *Proceedings of the fourth conference on email and anti-spam*. CEAS.
Settles, B. (2010). *Active learning literature survey*. Madison: University of Wisconsin.
Settles, B. (2011). From theories to queries: Active learning in practice. *Active Learning and Experimental Design, W*, 1–18.
Settles, B. (2011). Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1467–1478). Association for Computational Linguistics.
Settles, B., & Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1070–1079). Association for Computational Linguistics.
Smailović, J., Grčar, M., & Žnidaršič, M. (2012). Sentiment analysis on tweets in a financial domain. In *Proceedings of the fourth Jožef Stefan international postgraduate school students conference* (pp. 169–175). Jožef Stefan International Postgraduate School.
Smailović, J., Grčar, M., Lavrač, N., & Žnidaršič, M. (2013). Predictive sentiment analysis of tweets: A stock market application. In *Proceedings of human–computer interaction and knowledge discovery in complex, unstructured, big data* (pp. 77–88). Springer.
Talia, D., Trunfio, P., & Verta, O. (2005). Weka4WS: A WSRF-enabled Weka toolkit for distributed data mining on grids. In A. Jorge, L. Torgo, P. Brazdil, R. Camacho, & J. Gama (Eds.), *Proceedings of PKDD* (pp. 309–320). Springer.
Taylor, I., Shields, M., Wang, I., & Harrison, A. (2007). The Triana workflow environment: Architecture and applications. *Workflows for e-Science, 1*, 320–339.

Thelwall, M., Buckley, K., & Paltoglou, G. (2011). Sentiment in twitter events. *Journal of the American Society for Information Science and Technology, 62*, 406–418.
Turney, P. D. (2002). Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 417–424). Association for Computational Linguistics.
Vanschoren, J., & Blockeel, H. (2009). A community-based platform for machine learning experimentation. In W. Buntine, M. Grobelnik, D. Mladenič, & J. Shawe-Taylor (Eds.), *Proceedings of machine learning and knowledge discovery in databases* (pp. 750–754). Springer.
Vapnik, V. (1995). *The nature of statistical learning theory*. Springer.
Vapnik, V. N. (1998). *Statistical learning theory*. Wiley.
Wang, P., Zhang, P., & Guo, L. (2012). Mining multi-label data streams using ensemble-based active learning. In *Proceedings of the 2012 SIAM international conference on data mining* (pp. 1131–1140). SDM.
Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques* (3rd ed.). Amsterdam: Morgan Kaufman.
Zhu, X., Zhang, P., Lin, X., & Shi, Y. (2007). Active learning from data streams. In *Proceedings of the seventh IEEE international conference on data mining, ICDM* (pp. 757–762). IEEE.
Zhu, X., Zhang, P., Lin, X., & Shi, Y. (2010). Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 40*, 1607–1621.
Žliobaitė, I., Bifet, A., Pfahringer, B., & Holmes, G. (2011). Active learning with evolving streaming data. In *Proceedings of machine learning and knowledge discovery in databases, ECML/PKDD* (pp. 597–612). Springer.