A Visual Programming Approach to Big Data Analytics

Christian Bockermann

Artificial Intelligence Group, Computer Science,
Technische Universität Dortmund, Germany
christian.bockermann@cs.tu-dortmund.de
http://www-ai.cs.tu-dortmund.de/

Abstract. Data processing and analysis has become a major task in a lot of application domains. Most tools for defining analytical processes lack a user oriented interface – especially when it comes to Big Data analytics.

In this work we propose an abstraction layer for process design that enables domain experts to define their processes at an abstract level that matches their expertise. Based on that, we investigate the use of machine learning to provide gesture recognition on input devices like tablets to provide these experts with a intuitive environment for process design.

1 Introduction

We face today massive amounts of data in various application domains: data collected in e-commerce, internet transactions, smart cities with thousands of sensors or sophisticated experiments in science. *Big Data* is often used as a synonym for the continuously generated amounts of information from heterogeneous sources. A key to Big Data is to provide analytical processes that allow for the extraction of any kind of knowledge from that data, be it by realtime processing or batch jobs that will scan data archives for useful patterns.

For addressing the analysis of data at large scale, a number of projects have been proposed – most notably perhaps Google's Map&Reduce framework for boosting batch computation along thousands of nodes [9,1]. In the area of realtime processing, stream oriented systems like *Storm* [10] or *S4* [18] have emerged, which provide low-level means for implementing analysis processes tailored towards a specific application domain. A survey of different state-of-the-art frameworks for Big Data approaches is given in [11].

The existing frameworks provide a code-level entry to the processing of streaming data. Programmers can provide custom code and build *streaming applications* at the programming level. For a lot of application domains, this leaves a huge gap between the processing power provided by the platforms, which is only accessible at the code-level and the high-level view of domain experts, whose focus is on their application specific problem. From the perspective of a domain expert (or *end user*) it is crucial to hide as much of the implementation level as possible. For example, a physicist is interested in designing a data analysis workflow,

A. Marcus (Ed.): DUXU 2014, Part II, LNCS 8518, pp. 393–404, 2014.

without having to to deal with the particularities of low-level code. This poses questions such as:

1. What level of abstraction is required to enable domain experts to use these large scale frameworks?
2. Can we find representations that allow for rapid prototyping by end-users whilst keeping the power of modern cluster systems?
3. What are reasonable interfaces to enable end-users to specify their analytical processes in the most intuitive and comprehensible ways?

To bridge the gap between the implementation and design level, we investigate the use of a *process design* layer – a higher-level abstraction to model *streaming applications* using simple elements and existing building blocks. This layer is independent from the underlying stream processing platform and allows execution of the designed process on various frameworks. As a second step, we propose the *sketch level*, which facilitates process design in an artistic way by interactive drawing on two-dimensional surfaces (e.g. tablets). Using machine learning for gesture recognition allows for individually adjusting the interaction to the process designer. Figure 1 shows the different levels of application development and the additional layers *process design* and *sketch level*.

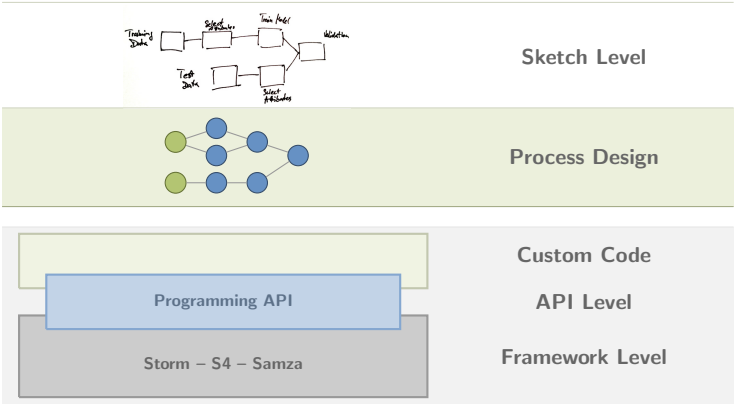

Fig. 1. The gap between process design and implementation. The *framework* to *custom code* levels exist in current platforms. This work investigates a proper process design level and provides a new *sketch level* for interactive creation of data flow processes.

The rest of the paper is structured as follows: In Section 2 we review the current landscape of related works and approaches in data processing. Next we present our process abstraction for streaming applications derived from the various existing approaches, providing a process design level for domain experts in Section 3. Based on the process representations we investigate human interaction with tablet devices to provide an interactive environment for process design

using machine learning in Section 5. In Section 6 we discuss the evaluation of the machine learning algorithms for the user interaction part. Finally, we conclude our findings in Section 7.

2 Problem and Related Work

As data analysis and processing have become a key task in various application domains, the need for tools providing such analysis capabilities to domain experts has emerged. In recent years, such tools have been developed for small to medium sized data and have been equipped with interactive visual components for process design. Examples for such (statistical analysis) tools are *RapidMiner*, SPSS, the *R* project, MatLab or the KNIME workbench [17,19,16,4]. These tools mainly focused on single node batch processing of data.

The growth of data volumes has forged a paradigm shift to massive parallelism and distributed computation. A major step towards *Big Data* has been Google's remake of the Map&Reduce paradim [9], from which the open source Apache Hadoop project has emerged [1]. In consequence various languages have been proposed to implement data processing on the Map&Reduce framework, e.g. *Pig* [2]. With the data *velocity* growing along, batch processing such as provided by the Map&Reduce framework is becoming a bottleneck to realtime analytics. As a consequence a number of *general purpose streaming platforms* have emerged, which allow for processing of streaming data in real time [18,10,5]. These frameworks provide low-level programming APIs for implementing custom streaming applications that fit the data analysis task at hand. Some of these platforms use configuration languages like XML [18,20] or domain specific languages (DSL) to let users specify streaming programs [13]. However, these approaches still are situated close to the implementation level.

With the low-level access to streaming architectures, it requires a high degree of expertise to make good use of these streaming capabilities in application domains that are not centered around IT. As an example, data in physics (e.g. telescope data analysis [3]) perfectly matches the streaming scenario, but physicists tend to lack the time to expertise in low-level streaming architectures. Their level of abstraction is to think about the information flow required for the processing of their experiment data and not the particularities to implement the analysis on low-level means of a streaming platform.

Within this work, we propose the use of interactive gestures on top of a high level stream definition language provided by [5] to enable users to *draw* the processes they require for their specific application. A taxonomy of gestures for user interaction has been surveyed by Karam and Schraefel in [14]. The use of machine learning for gesture detection as used in this paper is similar to the *handwriting recognition* problem, for which several algorithms – especially neural networks – have been proposed [15]. [6] further investigates different feature representations for handwriting recognition.

3 Abstraction of Data Stream Process Design

A natural perception of data stream processing is the modeling of data flows by means of a graph. Such a graph contains sources of data that continuously emit items which are processed by connected nodes that do some actual computation on the items. A very simple instance of such a graph is shown in Figure 2: A single source emits messages m_i, which are processed by some extracting node. This extractor in turn emits items t_i that are consumed by a counting node and the counts may be pushed to some other consumer.

Essentially two elements need to be present: a data source element and an element that defines the processing of items emitted by the source. In addition, a common representation for the data items to be processed is required. With different terminologies, these elements are present in all the streaming platforms. As an example, within the *Storm* framework, sources are referred to as *Spouts* and processing nodes are called *Bolts*. As contrast – in the S4 system [18], processing nodes are simply *Processing Elements*, but have the same semantical meaning.

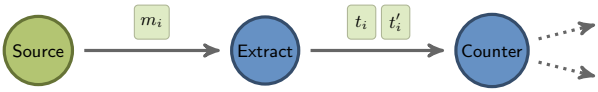

Fig. 2. A data flow graph for counting elements extracted from a stream of messages

3.1 Streaming Applications as Data Flow Graphs

The simple notion of sources, items and processing nodes, allows for defining *streaming applications* by means of connected components within a graph. As such graphs are the application structure in all modern streaming platforms, we proposed a unified modeling language for such application graphs within the *streams* framework [5]. It is an XML based definition language that allows for specifying data sources (*streams*) and processing nodes (*processes*) made up from existing functions to build data flow graphs that represent a streaming application. Figure 3 shows an example configuration for a streaming application defined in the XML dialect of the *streams* framework.

```
<application>
   <stream id="dataStream" url="file:/path/to/data.csv" />
   <process input="dataStream">
      ...
   </process>
</application>
```

Fig. 3. An example XML definition of a simple application that consists of a `stream` element and a single `process` consuming that stream

The functional units that a `process` is be made of within *streams* are so called *processors*. Processors are fine grained functions that are provided to the domain experts as the lowest building blocks and are executed by a process one after the other. Figure 4 shows the flow of data that is implied by the *streams* abstraction layer. By providing a user with a set of *processor* functions, this allows for users to create streaming applications by orchestrating the appropriate XML definitions. As these definitions are located at an abstract level, the focus is on the pure design of the data flow – without any notion of the underlying platform.

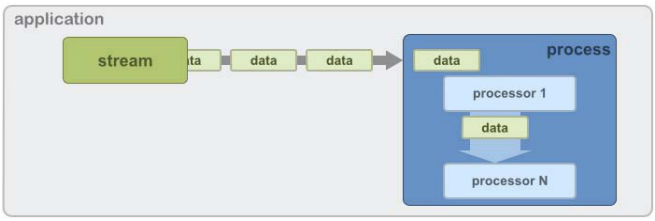

Fig. 4. A simple visualization of the data flow that has been specified in the XML example. The `process` element contains *processor* functions that are applied to each data item in order.

3.2 An Abstract Process Design Layer

Applications in *streams* can then be executed or deployed on the *streams runtime* or being compiled into programs that run on the *Storm* platform. This provides a level of abstraction for process design without focusing on the exact platform that the application is to be deployed on. By providing a variety of generic *processor* elements, users are enabled to create applications from existing building blocks. In addition, it is easy for experienced programmers to add additional application specific processor implementations that can be used by application designers (domain experts).

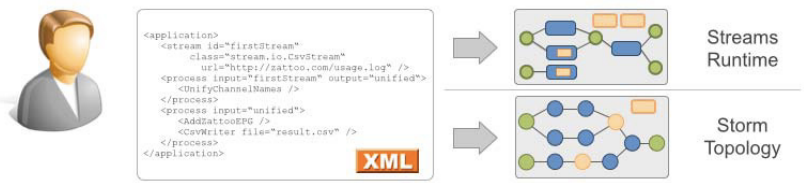

Fig. 5. Using an XML dialect for process design with the *streams* abstraction layer. The specified application can then deployed on different streaming platforms.

4 Deriving Stream Applications from User Sketches

The *process design* level provides an abstraction for elements that constitute streaming applications. For a further step we look into using this level of abstraction to provide an intuitive interface to process design by means of user sketches: Instead of letting the user specify her application using XML syntax, we use machine learning to derive processes from the user painting sketches on an interactive 2-dimensional surface. This further liberates the domain expert from technical details and fosters focusing on the particular problem at hand.

Starting with the representation of streaming applications as data flow graphs, these can easily be mapped to sketches on a paper sheet or drawing board. A simple visualization of such graphs has already been shown in Figure 2. In the following we are interested in reverting that process and deriving the XML process definitions from the sketches. Figure 6 shows a sketch of a process and its translation into a data flow graph that is backed by an XML definition.

In the following we will first formalize the interactive sketch pad area and the user actions that are obtained from that.

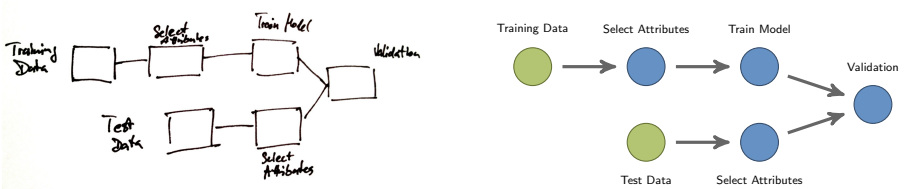

Fig. 6. A sketch of a simple streaming application and its corresponding graph visualization which is backed by the XML definition

4.1 Drawing Sketches and Gestures

The challenge we are looking at is how to derive the streaming application from the interactions the user performs for while sketching her application on a tablet PC or smart-board device. As the basis for our work we regard the user interactions to be performed on a 2-dimensional area, such as a touch pad, an interactive white board or a screen. The area can be interacted with by the use of a pointer, a finger or a pen. To formalize this, we define the notion of a *sketch pad* as follows:

Definition 1. *Let $w, h \in \mathbb{R}$ with $w, h > 0$. A* sketch pad *$P = [0, w] \times [0, h]$ is a 2-dimensional space with width w and height h. A user interaction with P is a sequence of points $(x_i, y_i, t_i) \in P \times T$ that result from the user touching the pad. Each point consists of coordinates x_i and y_i and a timestamp t_i of the time at which the interaction took place.*

The resolution of coordinate system of P and the time T is dependent on the device that is used for user interaction. Usually a time resolution in the order of

milliseconds is an adequate granularity. If we have a closer look at the creation of the sketches of Figure 6 it is obvious that they have been drawn with a sequence of moves of a pen. If we leave away the labels of the sketches, then Figure 7 shows different stages of the drawing that eventually lead to the sketch of the process as shown in Figure 6.

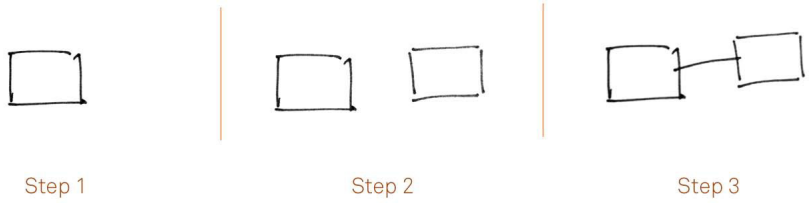

Fig. 7. Stepwise creation of a process sketch

The dissection of the drawing into distinct stages is one of the key parts for the process designer approach as each distinct stage reflects a single *gesture*. A simple way to break down the drawing into a sequence of moves is define a move as a single pen stroke. This leads to the following definition of a *gesture*:

Definition 2. *A* move *is a sequence of points that starts with the touch down of a pen and ends with the lift up of the pen. For a given time interval* τ*, a* gesture *is sequence of moves* $\langle m_0, \ldots, m_l \rangle$ *such that the time between the end of move* m_i *and the start of move* m_{i+1} *is less or equal to* τ*.*

With that definition, a *gesture* is a connected set of moves that leads to distinct parts of the drawing. For example, each of the two boxes of the last step of Figure 7 is the result of a single gesture. In case of the boxes, these gestures consist of multiple moves[1], whereas the line that connects the two boxes is the result of a gesture of only a single move. Another consequence of this definition is, that we can now regard a sketch simply as a sequence of gestures, each of which relates to an element of the sketch. The task of handwriting recognition, which is closely related to our gesture detection problem, inherits the problem of *segmentation*, i.e. a scanned writing needs to be dissected into single characters pictograms, which are then classified into letters. In the gesture detection, we solve this by splitting the interactive recording of gestures over time using the above definition.

4.2 From Gestures to Application Elements

Based on the dissection of figures into gestures, we now need to define a mapping of gestures to the elements that the application may contain eventually. That is, the elements a user is "drawing" are *stream* or *processor* elements and edges

[1] We assume that the boxes have been drawn with multiple strokes. Of course it is also possible to draw the same boxes with a single stroke only.

connecting these elements. We are however not limited to only map gestures to elements of the application – we may also be interested in providing gestures that perform actions on these elements. As an example, we define a *delete* gesture, which provides a convenient way to remove elements from an application sketch. Figure 8 shows the defined gestures that we used for this paper.

Fig. 8. Gestures defined for user interaction and process design

5 Machine Learning for Gesture Recognition

The gestures are derived from the user interactions based on the gesture definition in Section 4. Given a predefined set of known gestures, such as listed in Figure 8, the mapping of gestures to process elements can be viewed as a multi-class classification task: Input to the classification is a feature representation of the gesture and the output (class label) will be the correct application element or action that is associated with the gesture. In the following section we will outline a basic feature representation for gesture prediction and extend this with some extra information that can be derived from the gesture context.

5.1 Feature Representation for Gestures

The general idea for representing gestures as a feature vector reclines from the MNIST handwriting recognition [15]. This popular data set is related to the task of identifying digits from handwritten graphics, e.g. ZIP codes that have been scanned from letters or post cards. Figure 9 shows sample digits of the MNIST data set.

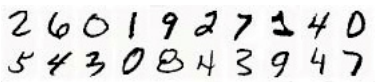

Fig. 9. Some digits of the MNIST handwritten dataset

The digits are represented by a grayscale bitmap image of some squared size like 28×28 pixels. Each pixel holds a gray color value. The feature vector extracted from the digit images is a sparse vector of length $p = 28 \cdot 28 = 784$ where each attribute represents a pixel of the bitmap image. Alternative weighting schemes are for example the weighting of each pixel by its euclidean distance to the center [6].

Recording Features from Gestures. An important advantage over the handwriting recognition task as provided by the MNIST data is the interactivity of the recorded gestures in the use case at hand: In addition to the points a user touches when performing a gesture, we are able to obtain additional information such as the order in wich the gesture is drawn as well as speed and timing information of the gesture. By recording *pen-up* events we are also provided with any interruptions of the gesture, which provides the number of strokes/moves a gesture is build upon. With this information we are able to extract much more fine grained feature vectors from each of the users gestures than only relying on the bitmap features as proposed in the MNIST dataset.

Based on the recordings of user interactions, we mapped each gesture onto a square area of size 16×16 and derived the following features for each of the gestures:

1. *per-pixel feature*: euclidean distance of the pixel to the center,
2. *gesture duration*: time in milliseconds of the user interaction,
3. *height*, *width*: original height, width of the gesture interaction (in pixels),
4. *numberOfStrokes*: the number of strokes the gesture consists of,
5. *scale:x*, *scale:y*: factors used to scale the gesture to the 16×16 square.

With the gesture area size of 16 pixels and the additional features enlisted, we end up with 295 numerical base features for each of the recorded gestures (289 pixel-features plus the 6 additional features). The recordings themselves are performed at the resolution of the sketch pad at hand. This allows for variations of the feature extraction by means of different scalings (instead of 16×16 pixels). In addition, the weight of each pixel feature can be based on different objectives, such as the center distance (see above), the time of the pixel recording or a simple binary value (e.g. 1.0 for pixels that belong to the gesture, and 0.0 for the other pixels).

5.2 Adding Context Features to Gesture Examples

While recording gestures during user interaction, a few additional features can be derived from the *interaction context*. The *interaction context* is given by the set of elements that have been defined based upon previous gestures. A straight forward example are the start- and end-points of gestures. Figure 10 shows the *Connection* gesture (orange stroke) that starts within a source element and ends in a target element. The start and end elements can be encoded into the feature vector using binominal features and may not only refer to the exact element level, but may also result in the *type* of elements from which a gesture starts or on which a gesture ends. As an example, the *Connection* gesture in Figure 10 could be enriched with features *source:type=process* and *target:type=process*. Other context based features may be derived from application elements that are covered by the area of a gesture or elements that appear near the area of a user gesture.

Fig. 10. Gesture that starts in one element and ends in another one

6 Evaluation

The goal of this work is to raise the level of application design to ease the definition of streaming applications for domain experts. For this we introduced the *process design* layer as a first abstraction and the *sketch layer* on top of that. We to evaluate these two parts separately: The process design using XML and the use of machine learning for interactive sketching.

6.1 User Experiences for Process Design

For collecting a sentiment for the process design level provided by the *streams* abstraction, we investigated the use of *streams* in collaboration with several projects: A group of physicists working on the FACT telescope data analysis; the ViSTA-TV EU project [8]; the EU project INSIGHT [7] and finally within a workshop for high-school students. In all of these areas, the users ingested the abstract data flow design view and were quickly able to specify their stream processes using the XML notation. The decoupling of code and design led to multiple contributions of processor implementations and enabled the physicists to combine these processors into several experiments simply by altering the XML.

Within the ViSTA-TV and the INSIGHT projects, the *streams* XML was used to integrate processes of various groups within a central application. In addition, the flexibility of the XML configuration led to new streaming applications within a completely different domain for the DEBS challenge 2013 [12] that was easily put together by members of the INSIGHT project.

For the workshop with high-school students, the *streams* framework was used as a base environment for addressing a streaming task in video analysis (event detection) by the students. The high-level notion of process design enabled the students to perform various different experiments within only a 2-day workshop without any a-priori knowledge.

6.2 Evaluation of the Gesture Prediction

For evaluating the interactive sketching of processes, we implemented a *sketch app* for an Android tablet that provides a clean and simple sketch area. In a first step, we used this app to record user gestures that were then manually classified into one of the predefined gesture actions. For each of the five gestures we created 60 samples and extracted feature vectors from these 300 gestures. The feature vectures were normalized and used for training a classifier for gesture prediction.

As we were especially interested in the required resolution of the per-pixel features, we re-created the example set with different per-pixel resolutions by

scaling the gestures to formats of 12×12, 16, 24×24 and 32×32 pixels. For estimating the quality of the features, we used a fixed learning scheme – a SVM classifier with a linear kernel ($C = 1000.0$). The best prediction results were achieved using per-pixel features only for a gesture format of 12×12 pixels.

With the training data available, we investigated different classifiers and parameters for the gesture prediction using a *1-against-all* strategy. The following Table 1 enlists all the classifiers that have been tested on the data set using a 10-fold cross validation. In this evaluation we made use of all available feature (including context features) and used a resolution of 12×12 pixels with binary weights per pixel.

Table 1. Different classifiers tested for gesture prediction

Classifier	**Accuracy** (Variance)	**Worst Recall (Class)**
NeuralNet, learn rate 0.3, 100 cycles	**99.67% (± 1.00%)**	98.33% (operator)
Linear SVM, C=1000.0	99.67% (± 1.25%)	98.33% (delete)
Perceptron, learn rate 0.05	98.00% (± 2.67%)	93.33% (source)
Naive Bayes	93.67% (± 4.58%)	90.00% (source)
Naive Bayes (Kernel)	92.24% (± 5.83%)	86.44% (source)

The evaluation results of Table 1 summarize the overall classification rate. As this is a multi-class classification problem, we are especially interested in the worst class recall, which is highlighted in the last column. With an accuracy of *at least* 98.33%, the correct gesture is detected, which makes the *sketch app* usable without much correction required.

7 Summary and Future Work

We presented a two-step approach to the design of streaming applications for data processing. The versatility of the *streams* abstraction and its flexibility highly matched the experimental work flows in each of the groups and the data flow concept was easy to pick up by the involved people. Our prototype of a gesture based sketch application for process design provided high precision for mapping gestures to application elements using machine learning and provides and interactive and extendable approach for application design.

Future work will be focused on providing a more complete real world application designer based on the sketch concept and the investigation of additional machine learning tasks, e.g. for recommendations of processors and parameters based on previous usage.

Acknowledgements. This work is funded by the Deutsche Forschungsgemeinschaft within the CRC SFB-876 "Providing Information by Resource-Constrained Data Analysis", project A1. Further support was provided by the ViSTA-TV EU project, grant number 296126. We thank the physics department for valuable input and collaboration.

References

1. Apache Hadoop (2007), http://hadoop.apache.org/
2. Apache Pig (2008), http://pig.apache.org/
3. Anderhub, H., Backes, M., Biland, A., et al.: Design and operation of FACT – the first G-APD Cherenkov telescope. Journal of Instrumentation 8(06), P06008 (2013)
4. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., Wiswedel, B.: KNIME: The Konstanz Information Miner. In: Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007). Springer (2007)
5. Bockermann, C.: The streams framework (2012)
6. Chakraborty, B., Chakraborty, G.: A new feature extraction technique for on-line recognition of handwritten alphanumeric characters. Information Sciences 148(14), 55–70 (2002)
7. The INSIGHT Project Consortium. Intelligent Synthesis and Real-time Response using Massive Streaming of Heterogeneous Data (2012-2015), http://insight-ict.eu
8. The ViSTA-TV Project Consortium. ViSTA-TV – Video Stream Analysis for the IP-TV Industry (2012-2014), http://vista-tv.eu
9. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
10. Marz, N., et al.: Storm - distributed and fault-tolerant realtime computation (2013)
11. Fan, W., Bifet, A.: Mining big data: current status, and forecast to the future. SIGKDD Explorations 14(2), 1–5 (2012)
12. Gal, A., Keren, S., Sondak, M., Weidlich, M., Blom, H., Bockermann, C.: Grand challenge: the techniball system. In: Chakravarthy, S., Urban, S.D., Pietzuch, P., Rundensteiner, E.A. (eds.) DEBS, pp. 319–324. ACM (2013)
13. Yieldbot Group. Marceline – a Clojure DSL for Storm/Trident (2013)
14. Karam, M., Schraefel, M.C.: A taxonomy of gestures in human computer interactions. Technical report, University of Southampton (2005)
15. LeCun, Y., Cortes, C.: The MNIST Database, http://yann.lecun.com/exdb/mnist/
16. MATLAB. version 7.10.0 (R2010a). The MathWorks Inc., Natick (2010)
17. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: Yale: Rapid prototyping for complex data mining tasks. In: Ungar, L., Craven, M., Gunopulos, D., Eliassi-Rad, T. (eds.) KDD 2006: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 935–940. ACM, New York (2006)
18. Neumeyer, L., Robbins, B., Nair, A., Kesari, A.: S4: Distributed Stream Computing Platform. In: International Conference on Data Mining Workshops, CA, USA, pp. 170–177. IEEE Computer Society (2010)
19. R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2013)
20. Spring. Springframework reference manual 3.1 (2011)