Making Machine-Learning Applications for Time-Series Sensor Data Graphical and Interactive

SEUNGJUN KIM, DAN TASSE, and ANIND K. DEY, Carnegie Mellon University

The recent profusion of sensors has given consumers and researchers the ability to collect significant amounts of data. However, understanding sensor data can be a challenge, because it is voluminous, multi-sourced, and unintelligible. Nonetheless, intelligent systems, such as activity recognition, require pattern analysis of sensor data streams to produce compelling results; machine learning (ML) applications enable this type of analysis. However, the number of ML experts able to proficiently classify sensor data is limited, and there remains a lack of interactive, usable tools to help intermediate users perform this type of analysis. To learn which features these tools must support, we conducted interviews with intermediate users of ML and conducted two probe-based studies with a prototype ML and visual analytics system, Gimlets. Our system implements ML applications for sensor-based time-series data as a novel domain-specific prototype that integrates interactive visual analytic features into the ML pipeline. We identify future directions for usable ML systems based on sensor data that will enable intermediate users to build systems that have been prohibitively difficult.

Categories and Subject Descriptors: D.2.2. [**Design Tools and Techniques**]: User interface

General Terms: Human Factors

Additional Key Words and Phrases: Machine learning tools, visual analytic tools, big sensor data, wearable devices

ACM Reference format:
SeungJun Kim, Dan Tasse, and Anind K. Dey. 2017. Making Machine-Learning Applications for Time-Series Sensor Data Graphical and Interactive. *ACM Trans. Interact. Intell. Syst.* 7, 2, Article 8 (July 2017), 30 pages.
https://doi.org/10.1145/2983924

This research was supported by the National Science Foundation under Grant No. CCF-1029549, *in part* by Carnegie Mellon University's Technologies for Safe and Efficient Transportation, The National USDOT University Transportation Center for Safety (T-SET UTC) which is sponsored by the US Department of Transportation under Grant No. DTRT-13-G-UTC26, and by the Ministry of Trade, Industry and Energy (MOTIE) and Korea Institute for Advancement of Technology (KIAT) through the International Cooperative R&D program [N0001228, Development of UI/UX Technology to Overcome the Limitations of Wearable Device UIs].
This work is supported by the National Science Foundation, under grant CCF-1029549, by the US Department of Transportation, under grant DTRT13-G-UTC26, and by the Ministry of Trade, Industry and Energy and Korea Institute for Advancement of Technology, under grant N001228.
Authors' addresses: S. Kim, D. Tasse, and A. K. Dey, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania, United States, 15206; emails: sjunikim@cs.cmu.edu, dantasse@cmu.edu, anind@cs.cmu.edu.
Author's current address: S. Kim, Institute of Integrated Technology, GIST (Gwangju Institute of Science and Technology), 123 Cheomdangwagi-ro (Oryong-dong), Buk-gu, Gwangju, 61005, Republic of Korea.

https://doi.org/10.1145/2983924

1 INTRODUCTION

A plethora of sensors are currently available to technology users. These sensors are wearable and can be embedded in a physical environment. As a result, they generate big personal data that help researchers understand a person's behavioral patterns or track real-time states in cognition and attention. Smart phones, for example, now carry an array of sensors, such as accelerometers, gyroscopes, and ambient light sensors. Many people carry wearable sensors, such as pedometers, heart rate monitors, and skin conductivity sensors, as well. In addition, smart thermostats, security systems, and home automation systems, among others, generate useful data about homes and offices.

Unfortunately, these sensors only generate data; to build a variety of useful interactive applications, one often must build a system to classify these data. There are a growing number of HCI (Human Computer Interaction) applications available that rely on classification of sensor data patterns, particularly in the domains of health and sustainability. For example, sensor data have been used for activity recognition, in which a developer monitors multiple wearable sensors to determine if a user is performing an activity such as walking or bicycling in a given time period (e.g., (Bao and Intille 2004)). Activity recognition in particular has many healthcare applications, from fitness monitoring to eldercare support (Choudhury et al. 2006). Similarly, sensor data have been used to support gesture recognition, in which a developer might want to classify a short window of accelerometer and orientation data as one gesture or another to enable natural free-space interactions (Huang et al. 2009). Environmental sensors such as carbon dioxide, motion, and light sensors might be used to tell when a room is not in use so the lights can be turned off to conserve electricity (Lam et al. 2009). Finally, using physiological wearable devices, sensor data have been used to predict users' stress levels (Hong et al. 2012).

Machine learning (ML) provides one way to classify sensor data, but it generally requires expert knowledge, possessed by few, to process or analyze the data. The difficulties posed by sensor data are threefold. First, the data are voluminous; sensors often produce data multiple times per second, which can lead to difficulty in identifying which portions of data to classify. Second, sensor data are multi-sourced; one might have a number of different sensors, as in the activity recognition case above. Therefore, it can be difficult to identify and/or remember the important characteristics or features of the data for analysis. Third, the data are often unintelligible to the naked eye. Sensor data usually consist of time series of numbers, and humans cannot see and interpret raw time-series data as easily as images or text. In sum, these attributes complicate turning sensor data into features and applying standard ML models. As a result, even though more people have access to more data than ever before, it remains difficult for non-experts to process. Even computer scientists who are familiar with ML face difficulties unless they are ML experts.

Currently, users of ML classify sensor data in a number of ways. Expert users build up libraries of their own highly customized code, often implementing their own ML algorithms for specific tasks. Intermediate users have some experience with ML, often with graphical user interface (GUI) tools or libraries like Weka (Hall et al. 2009). Similarly to experts, these users often build models using these pre-built tools. However, without the deep knowledge and tools the experts have, intermediate users often experience delays and frustration. We aim to make classifying sensor data easier for these intermediate users, so they can seamlessly include classification in their applications. We focus not on experts, whose number is naturally limited but instead on the larger and growing group of intermediate users in the field of HCI and beyond.

In this article, we present recommendations and a vision for future usable ML tools for intermediate users. We conducted a 10-person contextual inquiry and two 10-person probe-based studies of ML users. From these, we generate an understanding of the process that users perform when building ML classifiers, and a set of recommended features to support pre-processing

of time-series sensor data and post-processing of the classifier results. In particular, we recommend mixed-initiative support through end-user interactive visualizations based on mouse input for finding and manipulating erroneous and outlier data, for identifying appropriate features and classifiers to use, and for comparing the results of different feature set-classifier combinations. We also recommend support for video, interactive visualization of data annotations, and quick creation of derived features. Finally, we recommend support for plug-ins for extensibility and structured guidance for moving through the model creation process, particularly for less-experienced users. We believe that tools with these features will enable intermediate users to build models that are currently prohibitively difficult and will allow advanced users to experiment more than they currently do. We start with an overview of previous work focused on making it easier to perform ML classification.

2 USABLE MACHINE LEARNING

While ML experts can understand computational learning theory in artificial intelligence and then create their own ML algorithms or analytic tools, there has also been significant work in helping intermediate ML users who want to build ML models as *domain experts* rather than *ML experts*. Crayons (Fails and Olsen 2003) allows users to create image-based models interactively by labeling interesting parts of an image by drawing on the image. CueFlik (Fogarty et al. 2008) allows users to customize their image searches by creating a model interactively, labeling images according to categories. Similarly, ReGroup (Amershi et al. 2012) allows users to create models of friends by interactively labeling friends based on group membership. These systems allow domain experts to interact with ML to solve particular problems in their domains.

However, none of these systems support working with big sensor data streams or address the three characteristics that differentiate sensor data classification from other classification problems. Therefore, in this study, we aim to allow intermediate users to build ML models for problems in the sensor data-mining domain.

More closely related to our sensor data-mining domain, a CAPpella (Dey et al. 2004) uses "programming by demonstration" to create binary, or two-class, models based on sensor data from user-generated test runs. Exemplar (Hartmann et al. 2007) promotes general-purpose use by allowing real-time demonstration and annotation to quickly build prototypes of sensor-based applications. EyePatch (Maynes-Aminzade et al. 2007) helps people build computer-vision-based classifiers by example. However, the programming-by-demonstration approach does not scale well to large amounts of sensor data. Often when trying to improve performance, users collect hundreds or thousands of instances of training data, as we see with today's proliferation of sensors that continuously produce time-series data.

Because of the large amount of data involved when classifying time-series sensor data, users often turn to Weka or another more general-purpose ML tool. Gestalt (Patel et al. 2010) offers further support in this direction by encapsulating a large portion of the ML implementation and analysis pipeline within the same application. However, the major steps within Gestalt (parsing, attribute generation, training, and testing) are implemented *in code written by the user*. Additionally, while Gestalt does not attempt to support any particular domain, it has been primarily demonstrated on handwriting recognition and sentiment analysis, which do not share the three problematic attributes of sensor data discussed above. LightSIDE (Mayfield and Rosé 2013) extends the Weka toolkit to natural language by adding text processing tools, but we are not aware of any ML tool that particularly helps users deal with the volume, multiple sources, and unintelligibility of sensor-based time-series data.

Data visualization is also important when analyzing time-series data, especially complex sensor data, because of the need for exploration. To support this need, interaction techniques and

analytical methods are required as well (Aigner et al. 2011). Sometimes it is not clear what model should be built and what features to use until a human looks through the data to identify any patterns or correlations to generate or test hypotheses. Also, visual analytics allow users to combine visualization and statistical methods iteratively to enhance their mutual benefits (Keim et al. 2008). For example, KronoMiner (Zhao et al. 2011b) and ChronoLenses (Zhao et al. 2011a) support users in complex exploratory visual analysis by letting them transform and combine time series.

Gimlets, which we highlight in this study, is a domain-specific usable ML tool whose visual analytics are richest for user interactivity with big sensor data streams. The aforementioned visual analytic tools (e.g., KronoMiner, ChronoLenses, etc.) provide richer features for time-series data but do not integrate ML pipelines, nor have they been evaluated for intermediate ML users who build ML models as domain experts.

ModelTracker (Amershi et al. 2015) offers noteworthy end-user interaction with ML, but its visual analytic functions are primarily to allow end-users to interact with ML evaluation results, such as model performance. In contrast, Gimlets' interactive visual analytic features help end-users inspect and pre-process raw sensor data streams within the ML pipeline.

We believe the current work includes a genuine and timely contribution to the HCI community, especially where applications include intelligent systems that incorporate a range of sensor data streams from personal tracking devices and the internet of things. In general, our work aims to make ML more accessible; specifically, our work is designed for intermediate users of ML in HCI domains who are struggling with time-series sensor data for easily testing prototype applications or conducting pattern analysis using ML techniques. Following Amershi et al. (2014), we ground the design of our domain-specific interactive ML tool in the user-centric design process. Our system, Gimlets, supports interactive visual analytic features for a wide variety of sensor data, which are usually collected at high sampling rates, into the ML pipeline, and aligns videos with time-series data. To meet the needs of intermediate users, interaction with the Gimlets interface is primarily based on mouse input.

Our work contributes to recent work in the field. Patel's Gestalt, the most similar prototype system to include many of our Gimlets' features, not only supports the entire ML process but also is designed based on user-centric implications (Patel et al. 2010). More specifically, Gestalt's Prospect provides an interactive visualization tool that helps users generate different configurations of features, classification algorithms, and evaluation techniques (Patel et al. 2011), similarly to the interactive visual analytic features in Gimlets. Gestalt's Hindsight also supports the process of capturing experimental history to help programmers reflect on what they have done and make better experimental decisions (Patel 2013), similarly to History Tree in Gimlets.

However, while Gestalt supports a general workflow of implementation, analysis, and easy transitions between the two for machine learning, Gimlets includes rich user interactivity that accounts for large amounts of pre-processing and visual analytic tasks for massive time-series sensor data. As a general-purpose development tool for ML applications, Gestalt is limited to properly visualizing relationships inherent to sequential data (e.g., time series) and lacks domain-specific tools, as outlined in Patel et al. (2010). Our system, Gimlets, fills these gaps by offering richer user-interactive visual-analytic support specifically for handling time-series sensor data in an ML pipeline.

Gimlets allows users to access visual analytic features over the entire ML pipeline, and hence it is fairly distinct from DejaVu or d.tools, whose visual analytic features for time-series data are limited to interactive computer vision or visual prototyping. DejaVu, an IDE (Integrated Development Environment) enhancement, enables programmers to monitor program data consistent with the frame-based pipeline of computer-vision programs (Kato et al. 2012). Despite its support for interactive camera-based programs and a canonical development workflow, DejaVu does not

Formative interviews (Contextual inquiry) → **Probe study 1** (Gimlets 1.0, Classification task) → **Probe study 2** (Gimlets 2.0, Inspection task)

Group A	
	Position (Research Area, Years using ML)
A1	PhD Student (HCI, 4)
A2	Systems Scientist (HCI, 4)
A3	Research Programmer (HCI, 6)
A4	Systems Scientist (HCI, 9)
A5	PhD Student (HCI, 1)
A6	PhD Student (HCI, 4)
A7	PhD Student (HCI, 3)
A8	PhD Student (Computer Vision, 5)
A9	PhD Student (Computer Vision, 5)
A10	PhD Student (Language Technology, 5)

Group B	
	Position (Research Area, Years using ML)
B1	PhD Student (Electrical and Computer Engineering, 4)
B2	PhD Student (Machine Learning, 7)
B3	Research Programmer (Learning Science, 4)
B4	PhD Student (Language Technology, 4)
B5	PhD Student (HCI, 6)
B6	PhD Student (HCI, 2)
B7	PhD Student (HCI, 6)
B8	PhD Student (HCI, 5)
B9	PhD Student (Embedded Development, 1)
B10	PhD Student (HCI, 4)

Group C	
	Position (Research Area, Years using ML)
C1	PhD Student (HCI, 7)
C2	PhD Student (HCI, 4)
C3	Senior Researcher (Electrical Engineering, 15)
C4	Senior Researcher (AI + HCI, 13)
C5	Senior Researcher (Electrical and Computer Engineering, 15)
C6	PhD Student (HCI, 5)
C7	Senior Research Programmer (Computer Science, 3)
C8	PhD Student (HCI, 3)
C9	PhD Student (Language Technology, 5)
C10	PhD Student (Language Technology, 4)

Fig. 1. Participants in each user study.

integrate the ML pipeline. Similarly, d.tools offers an interactive-design-centered approach to prototyping information applications (Hartmann et al. 2006). However, although it links video to time-series information, the toolkit is primarily a visual design tool, not an ML tool. Hartmann et al.'s study group was composed of participants with general "design" experience, so neither their system nor their evaluation focused on an ML pipeline. Our system successfully integrates useful visual analytic features into an ML pipeline. We demonstrated the advantages of our approach by studying intermediate ML users.

3 UNDERSTANDING DOMAIN-SPECIFIC DEMANDS—FORMATIVE INTERVIEWS (GROUP A)

Our first goal was to better understand the difficulties involved in modeling sensor-based time series data and to understand where there is room for improvement in current practices. Patel et al. have extensively studied how people use statistical ML for general datasets (Patel et al. 2008a, 2008b]. However, their participants used many types of data, not just sensor data. Because we are particularly interested in sensor data, we conducted our own interviews to confirm and extend their results to sensor data.

For our Formative Interviews (Group A), we recruited 10 ML end-users with a few years of ML experience from the computer science department at our university. (ML years of experience: $M = 4.6$, $SD = 2.1$; see Figure 1 above). The computer science department includes multidisciplinary programs for robotics, language technology, educational technology and applied learning science, and human-computer interaction. Because our goal is to open ML to more "intermediate" users, we recruited users who had had some experience with ML but had not implemented their own ML algorithms.

We used a retrospective contextual inquiry approach by asking participants to walk us through a recent project they had completed that involved ML classification. After aggregating and analyzing the transcripts of the discussions with our participants, we discovered common themes about the classification process. We then identified domain-specific difficulties that arose where additional tool support could make it substantially easier for ML users with limited experience to classify time-series sensor data. Each participant's workflow included slightly different steps; however, most users followed similar processes as described below.

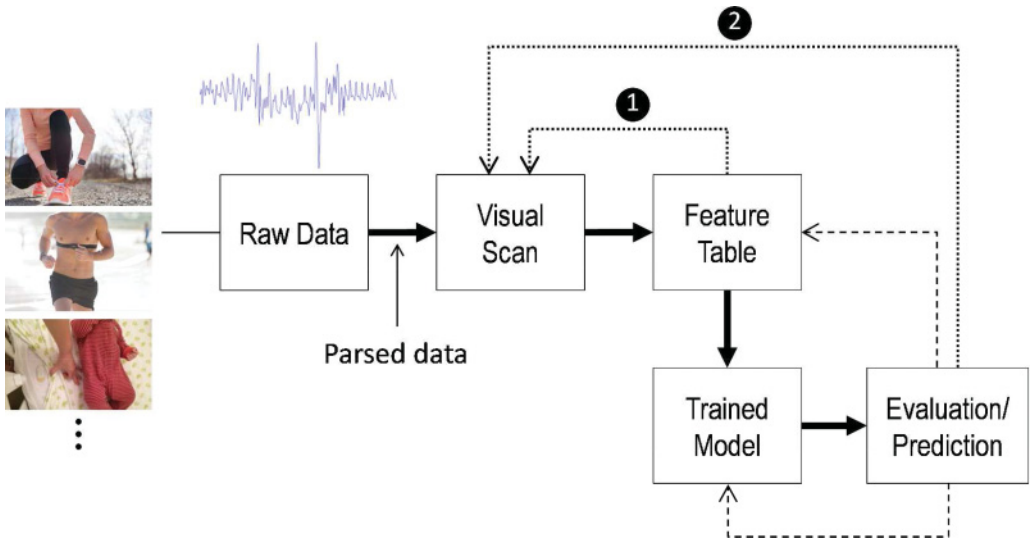


Fig. 2. ML process overview—main ML classification pipeline (solid arrows in bold) and iteration processes (dashed and dotted arrows) in participants' current practices.

3.1 Iterative Process Overview

We found that our participants began the model building process by visualizing and familiarizing themselves with their data. Next, they computed features for each instance to be labeled and passed those features to an ML algorithm that built models. Our participants then evaluated the accuracy of the model and repeated the process until they were satisfied with its accuracy. Even after the entire process was completed once, most of the tasks were repeated to improve or reason about the results in the evaluation/prediction step—the dashed and dotted arrows in Figure 2.

The classification pipeline that our participants generally followed is similar to the one depicted in Patel (2013) (i.e., raw data → parsed data → features → model → results). While Patel's ML pipeline includes an illustration of sensor-based activity recognition, the illustration does not particularly integrate the *Visual Scan* block and other important iterative processes that our participants pointed out (the dotted arrows, ❶ and ❷, in Figure 2). The focus of Patel's discussion is primarily on generic-purpose ML tools for ML applications such as digit recognition and web news classification and therefore is mostly about the other blocks and the solid and dashed arrows in Figure 2. We found that the iteration processes that are fed back to the Feature Table or Trained Model blocks in Figure 2 (the dashed arrows) were still important in our participants' sensor-based ML applications; however, many of our participants particularly highlighted their need for supporting the iteration tasks involving the Visual Scan process, which are more specific to the domain of our current interest, sensor data mining.

We found that our participants first built classifiers through an iterative process following similar workflow, which echoed Patel et al.'s (2013) findings using a generic-purpose ML tool for non-sensory data.

After performing an initial evaluation on their data, our participants expressed that they could not know if the model they produced was the most accurate model possible, so they would try at least a few times to improve their results. They iterated primarily in two ways: by changing which features they computed or by adjusting the model that they trained. Sometimes model adjustment

involved changing parameters, such as the maximum depth of a decision tree. Other times, it involved using a different model entirely, like a support vector machine instead of a decision tree.

Our participants usually reported deciding what to change by looking at the results of previous models. A4 examined the confusion matrix, a table visualizing how many instances in each class were classified correctly or incorrectly with regard to class, to see where common misclassifications occurred. A1 viewed graphs of some misclassified instances to identify distinguishing characteristics or features and to delete instances with incorrect ground-truth labels. A8 generated a large number of features and then let a feature selection algorithm select a subset of them. He examined visualizations of selected features to see if the model was ignoring any particular data.

We saw a wide variation in the time that our participants spent on this *results exploration* and *model improvement* stage. Users producing prototypes or proofs of concept would often spend a few minutes or hours iterating, as they had no set performance goals. However, A1 was designing a tool for production use, so she spent months iterating to improve classification accuracy up to 99%.

Many participants reported that they had trouble organizing the results of these iterations. A1 in particular had trained many different models with many different feature sets, and even participants like A6, who iterated much less, reported difficulties knowing which models they had already tried and what their accuracies were. Participant A8 reported that sometimes the models take a long time to train and classify, so he wanted to save the output of all of them, since rerunning them would take a long time. A8 reported encoding information about the models he built in the names of the output files (e.g., car_1_nfilt_20_iter_1_alpha_0.001.mat); later he had trouble determining when he had built each model or what the accuracy of each model was. Some of Patel's participants reported using similar schemes to keep track of their models, so we know that this is a significant problem in using ML. Many users could benefit from improved organization of their model building configurations and results. These examples confirm that history visualization is important to make model comparison easier in all ML domains, including our domain of time-series sensor data.

Next in this process of model improvement, participants who had extensive ML experience in sensor data mining (e.g., A2, A3, and A4) articulated the need for extending visual analytics and interactive end-user interfaces to support the iteration processes in loops ❶ and ❷ in Figure 2. To illustrate this process, we describe A2's project in more depth: A2 wanted to predict (from a smartphone and on-board diagnostic sensors' detection that a participant driver had been slowing down and was almost stopped at an intersection) whether sensor-detected driver and driving states (e.g., interruptible moments) would be a long session (e.g., a stop longer than 5s) or a short session (e.g., shorter than 5s). He wanted to design an in-car workload manager that regulated the flow of push notifications. First, he built a custom Visual C++ program to visualize multiple sensor data streams sequentially, so he could see examples of interruptible time-windows of various durations and parse the data correctly into instances. Next, he wrote another C++ program to generate a feature table, where each instance contained data from the 10s prior to multiple sensors starting to detect the driver's interruptible moments, and the class values represented short, medium, and long sessions. He built a random forest model, because he had used the classifier successfully for a similar binary classification problem; it had run efficiently on large datasets such as his sensor data. He then evaluated the accuracy of his model and compared it to naïve Bayes, which can be trained efficiently in a supervised learning setting. Because his model was not significantly better than naïve Bayes, he experimented with feature selection to reduce his feature set using the information gain metric. He also experimented with other models, including decision trees. After building a few different models over the course of a few days, he still was unable to outperform naïve Bayes. He started over by visualizing and inspecting the raw data streams again. He found

data values that could be potentially anomalous, but he needed to review black-box videos during data collection and perform a sort of confirmation task about the normal operation of sensors and confirm that there were no abnormal events and then modified his programs to appropriately process those data, that is, either remove or interpolate the data along with adjacent data points. As he was still unsure, he set new class criteria for session duration (e.g., longer vs. shorter than 10s), segmented time-series windows again, and then generated a new set of features, all of which significant time. He reported that model evaluation with a new feature table was relatively easy, because he already knew the series of models to test again and which corresponding evaluation approaches to take; however, he was still unable to get the expected model performance, so he started over again from the visual inspection of raw sensor data streams.

In this work therefore, we extend visual analytic features for sensor-based ML applications to support the more domain-specific iteration processes (i.e., ❶ and ❷ loops in Figure 2) that are hardly addressed by other ML tools (e.g., ModelTracker (Amershi et al. 2015), which mostly explores interactive visualizations of the results of ML model evaluation).

3.2 Raw Sensor Data and Real-World Contexts

In discussing their recent classification tasks, our participants began by discussing data that they had already gathered. They stated that they often wished to visualize that data. The visualizations they created usually took the form of simple line charts that are ubiquitous and easy to comprehend. Users stated that they did this for multiple reasons: to examine the data for noise or errors, to understand the data, and to formulate hypotheses about the data. A7 said that he wanted to "familiarize himself with the data" and "make sure the data isn't crazy." He was segmenting smartphone usage logs, which someone else had collected, into sessions that he could analyze further, so he wanted to make sure that he was performing this segmentation correctly.

Researchers may always open raw data files, but in the case of non-sensory data, they do not necessarily have to expend extra effort to make sense of the data, because it is relatively more intelligible to the naked eye (e.g., classifying images), as compared to the case of sensory data; therefore, extending the finding of the importance of data visualization (Patel et al. 2013), we especially emphasize the importance of *end-user interactive* visualization of sensor data along with *real-world contexts during data collection*.

Eight of our 10 Group A participants mentioned a need to create visualizations, particularly of raw sensor data. A3 created simple scatterplots from his sensor data and realized that much of the data at the beginning and end of each sample was noisy. A1 and A7, on the other hand, built up customized interactive tools just to look through their raw data. A1's took a long time to build, while A7's tool lacked features and frustrated him as he tried to look through gigabytes of phone usage data to identify which apps a phone user was using.

In addition, participants mentioned visualizing many other parts of the process as well. After creating a feature table, A6 and A7 mentioned looking at graphs of each feature individually to determine whether that feature alone could distinguish between classes. If it could, then they would include the feature in their model; if not, then they might or might not include it (as the feature might still be useful in combination with other features). A8 and A9 mentioned visualizing precision and recall curves of different models to see how each model handled the tradeoffs between different kinds of errors, and A9 showed us even more detailed three-dimensional visualizations of results.

Visualization offers an important opportunity to add effective end-user interaction with ML (Amershi 2012). In our work, we additionally emphasize that end-user *interactive* interfaces should help people understand how and why the current data streams look as they do, enable them to select time segments to inspect, and then enable them to process corresponding streams right there in

the same view. This interactivity should also be accessible over the entire ML pipeline. In practice, our participants wanted new ML tools to include richer visual analytic features, which particularly better supported sensor data mining. As A2 and A4 emphasized, it was important to "know real-world contexts during (raw) sensor data collection." In particular, sensor data streams from wearable devices can include potential noise or erroneous data (when deployed in field settings such as the home, in vehicles, in the gym, etc.) which makes data interpretation difficult. As A2 mentioned, for example, chest-belt-type physiological devices generate artificially high heart rate readings (higher than 150 bpm) when they are loosely fastened around the body trunk. As human subjects commonly slouch while walking or driving, it can be hard to confirm whether readings indicate a normal high heart rate caused by physical activity or mental workload or are just null data generated as a consequence of posture. Therefore, plotting raw data alone did not allow our participants to selectively inspect particular ranges of time-windows and validate corresponding sensor data streams in those ranges. Our participants needed to review videos of their human subjects during data collection before they could confirm whether to remove or process corresponding data streams. Based on our participants' comments, we highlight the domain-specific demands of comprehending sensor data streams along with "real-world contexts" during data collection to support users' visual inspection tasks without using extra tools.

3.3 Generating a Feature Table

After conducting their data parsing and inspection tasks, our participants generated features from their data to prepare to build ML models. Occasionally, they used their visualizations to select features to generate. More often, they used large sets of features that they had used previously or features they thought would be useful based on their domain knowledge. The outcome of feature extraction is a feature vector for each instance; together these vectors form the feature table. Feature generation was often manually performed using scripts in Python, Matlab, or other familiar high-level languages. The most common feature table representation was a comma-separated value (.csv) file, a plain text format where instances are listed one per line with commas separating each feature in the feature vector.

There exist domain-specific ML tools that help people to better interact with non-sensory data (e.g., ReGroup for on-demand group creation in social networks (Amershi et al. 2012)). In addition, there are rich interactive visual-analytic tools that account for large amounts of feature generation tasks for massive time-series data (e.g., ChronoLenses (Zhao et al. 2011a)). However, ML tools still lack the ability to integrate end-user interactivity for feature generation tasks within an ML pipeline, especially to support sensor data mining. As A2 commented, even when a user analyzes massive sensor data by increasing heap size of generic ML tools that also provide graphical user interfaces (e.g., Weka and its GUI), the user has to write a considerable number of custom scripts or use separate tools to derive new features and create a single feature table.

Another barrier that our intermediate ML users faced involved making sensor data usable at all by ML algorithms. Most ML software assumes that the user's data has already been processed into a feature table and allows the user to start building a model by importing such a table. As mentioned previously, however, sensor data are far more voluminous, multi-sourced, and unintelligible than data for textbook tasks like spam recognition or movie review analysis. For time-series sensor data, the data are not equivalent to the features, are not already segmented into instances, and, furthermore, cannot be represented in a simple table along with the features. Therefore, generating a feature table from time-series data is a significant task. Users must detect and remove noisy or erroneous data, segment their data into instances, and compute features from each instance. In addition, sometimes users must perform other steps to make their data usable, like parsing data from sensors' proprietary formats. Sometimes they must add intermediate data transformation

steps (e.g., computing the Fourier transformation of the data) to extract the features they want. These steps all contribute to a more complicated process.

Because the generation of a feature table is often an ad hoc process, it is often tedious, time consuming, and difficult to maintain, involving the creation of a complex Python, Java, or Matlab script. One user, A7, wrote over 5000 lines of Java code just to perform this first step; another, A2, commented that generating this big feature table can be "60% of the work" of the whole classification project. The features are often easy to compute individually, like means and medians of time series, but writing custom code to compute all of the features can add up to a lot of work. Even our most experienced user, A4, said that feature selection was a time-consuming process, even though he does not use many complex feature generation techniques.

To resolve this feature generation issue, we highlight the domain-specific demands of supporting time-series batch feature generation primarily based on mouse input without using separate tools or custom scripts for the task.

3.4 Model Training, Evaluation, and Iterative Segmentation of Time Windows

With the feature table, our participants generally continued by training a model using an ML algorithm provided by tools like Weka, Matlab, or Scikit-learn (Pedregosa et al. 2011) Those tools provide options for using classifiers such as decision trees, support vector machines, and naïve Bayes classifiers. Participants usually reported knowing which model they wanted to use or trying multiple models to see which would yield the best results. They would then evaluate the model using simple metrics like accuracy, correlation coefficient, or Cohen's Kappa by comparing their model's classification of instances to the instances' ground-truth labels.

Cross validation was the most common way to perform this evaluation. Sometimes participants used GUIs provided by their ML software, but more frequently they wrote custom scripts to interface with APIs due to GUI tool inflexibility (e.g., Weka). For example, A3 and A6 each wanted to use leave-one-person-out cross-validation instead of randomized cross-validation. This means that they wanted to train on all users except one, and then test on that one, instead of testing on a random subset of data or a subset based on values of one feature. To do this, they needed to write their own scripts to split their data into folds manually. As Patel et al. (2008b) also found, participants would sometimes use random cross-validation instead of per-participant, just because their software supported it. It may be more important for a sensor-data-focused tool to support per-participant cross-validation, because the data come from multiple sources and therefore may be difficult to segment manually. A6 also wanted to optimize his model to equalize the error rates so the number of false positives equaled the number of false negatives, but GUIs in most of existing ML software underdeveloped this optimization.

In addition, the process that our participants followed in building a model contained many sequential steps, which each depend on the previous step (Figure 2). Furthermore, many of our participants needed to introduce other steps, for example, dividing their data differently or copying their data. A2 created copies of his data, which allowed him to investigate the impact on classification of different window sizes. In one copy, he split his data into 3s windows and in another 10s windows. A3 created eight different train/test sets based on different window sizes and feature sets, while A7 had to split his data into different chunks based on varying-length phone use sessions. A4 and A8 reported adding an extra automated feature selection step to reduce their feature set from thousands to hundreds, which produced a faster and more efficient model building process. The combination of all of these steps turned the process into a multi-step pipeline, and, as A3 and A6 explained, changing one step in the pipeline made them rerun the feature table generation, model building, and everything else in the pipeline for multiple sets of data. As A3 said, if he has to rerun the pipeline 80 times, he has "80 times to get [part of the multi-step process] wrong."

To support these processes, we highlight the domain-specific demands of segmenting sensor data streams with iteratively defined time-windows, thereby helping users test model performance with differently grained features (e.g., sensor features segmented every one-second vs. every one-minute) and then run validations such as leave-one-sensor-out, leave-one-experiment-session-out, leave-one-person-out, and so on, which can be more useful for time-series sensor data.

4 GIMLETS SYSTEM DESIGN

Overall, we found that the problems with sensor data manifested themselves in four ways. It is difficult to turn sensor data into features, visualize and understand these data, manage the steps needed to classify sensor data, and organize the models that must be built. We hypothesized that addressing these four issues would greatly reduce the friction involved in building classifiers; thus, we developed a tool for usable ML of time-series data, Graphical Interactive Machine Learning, Especially for Time Series (Gimlets) (see Figure 3). Unlike general-purpose tools like Weka or Gestalt, Gimlets is domain specific. This study explores Gimlets to probe the value of various features that could be built into future tools. In this section, we describe the architecture and features of Gimlets.

4.1 Built on LightSIDE

We designed customizable Gimlets plug-ins for developers to easily incorporate new visual analytic features and end-user interaction schemes into the complex ML pipeline without necessarily modifying the other panels in Figure 3.

To accomplish this goal, we built Gimlets by modifying the open-source tool LightSIDE (Mayfield and Rosé 2013). LightSIDE, a Java application, was built using the Weka machine-learning library and functions as a GUI wrapper with functionality added to support classification and regression of text data. Example tasks that it supports include sentiment analysis and essay grading. Like Weka's GUI, it uses a panel-based design to encapsulate each task in the standard ML pipeline in its own panel. The most important panels allow users to generate feature tables, build models, and compare models to detect whether one is more statistically accurate than another. LightSIDE allows developers and end-users to create plug-ins for customizable parts of the pipeline, such as features to extract and ways to evaluate models.

It would be impractical to use LightSIDE on its own for our tasks on time-series sensor data, as the tools it provides are specifically focused on text data. For example, it allows the user to generate features based on unigram and bigram frequencies within the data. We retained the core workflow of building and evaluating models, but we added the following features.

In this study, we added a step to the LightSIDE pipeline, "Load and Preprocess Data" panel (Figure 3, front layer). Sensor data are quite complex and users often spend significant time wrangling their data into a large .csv file, as we found in our interviews. In addition, our participants repeatedly identified the ability to visualize raw data as a need. We customized LightSiDE plug-ins to accommodate these demands.

On the left pane of the panel, the user can load data in which each directory contains a single instance. Each instance may contain a .csv file, multiple .csv files, and/or time-series data in other formats. Users can then manipulate each time series (each column in a .csv file) separately. Scalar values can be included as well, as a one-line .csv file. On the center pane, we plugged in a multimedia player. This plug-in allows users to play and control videos to monitor real-world events during data collection. On the right pane, we included a large window for visualizations. As users load their data, they can view any time series as a standard line graph with Gimlets 1.0 (Figure 4) and richer visual-analytic features with Gimlets 2.0 (the right pane of Load and Preprocess Data panel in Figure 3). Users can display as many graphs of as many data series as they

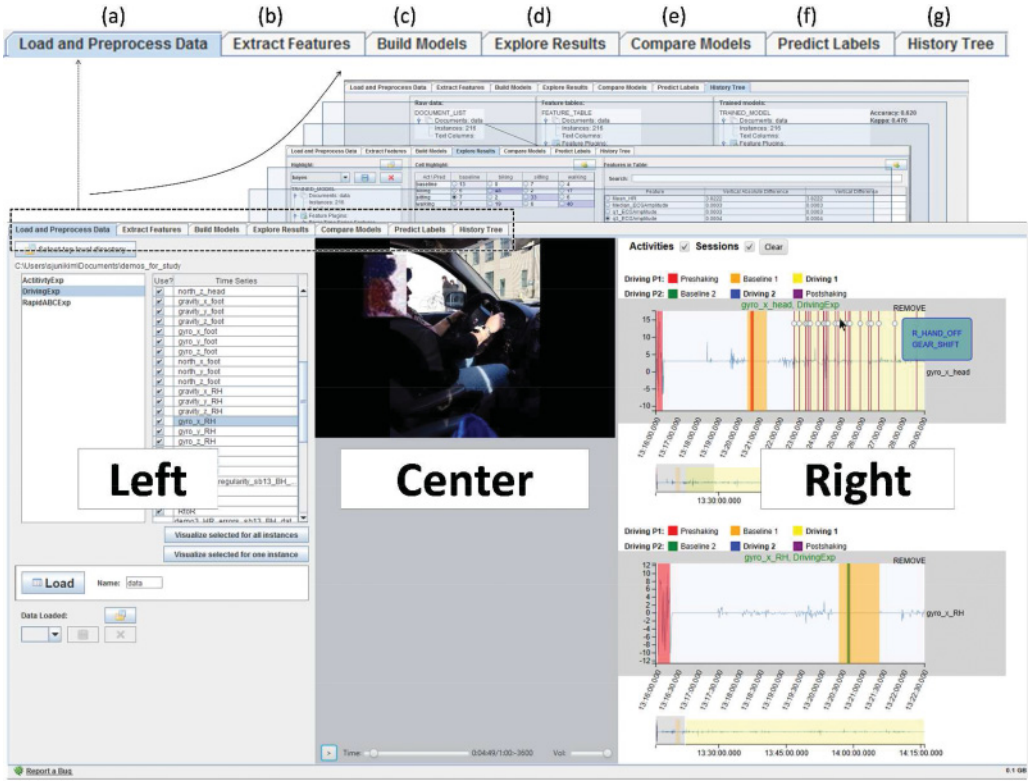


Fig. 3. Visual layouts for each step in Gimlets 2.0: (a) Load and Preprocess Data, (b) Extract Features, (c) Build Models, (d) Explore Results, (e) Compare Models, (f) Predict Labels, (g) History Tree.

choose, either for one instance at a time or across all instances (e.g., the breathing rate and skin temperature of one user or the heart rate of every user).

4.2 Interactive Visual Analytic Features in the Complex ML Pipeline

We designed Gimlets to support a range of interactive visual analytic features, which enable end-users to inspect, re-visualize, and process data primarily based on mouse input without relying on extra custom scripts. This design thereby helps intermediate users of ML who struggle with time-series sensor data and makes ML more accessible by streamlining the complex ML pipeline iteration processes.

First, we added the capability to create a new time series derived from a previous one with only two clicks (Figure 5). For example, a user could compute the Fourier transform or the derivative of a time series. Our participants reported already doing this, but adding the capability into our tool would save time and effort required to write custom code for every classification task. By default, we only included derivatives, though we enabled users to add more through LightSIDE's plug-in architecture.

Second, to help users create new features *in situ*, we developed an interactive feature: the *peak detector*. Because counting peaks can be useful with physiological data such as galvanic skin response—even though the definition of a "peak" might depend on the data—we added the capability to click on a graph to define what numeric value constitutes a peak (Figure 6, peak detector

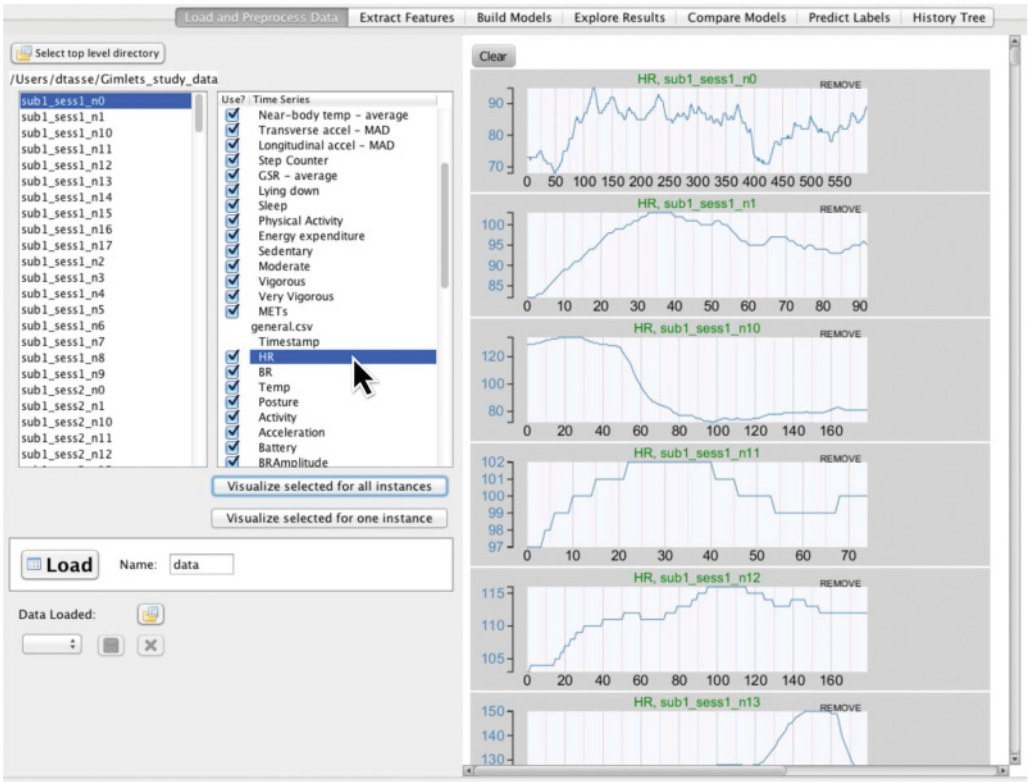


Fig. 4. A screenshot of Gimlets 1.0 with a standard line graph.

in the feature extraction panel). We developed this as an example of interactive features to elicit feedback on whether interactive feature generation could be useful in future tools.

Third, because feature table generation can be particularly problematic for sensor data, we aimed to make it easier in Gimlets. We replaced the text-specific feature extraction tools built into LightSIDE with time series-specific tools. By selecting values from a list, users can include commonly used features for each time series such as *mean, standard deviation, median* and *quartiles*, and *histogram* features (Figure 6, right side of the panel). We aim to keep this extensible through plug-ins, so if users often extract the same features from their data, they can write a plug-in once to extract that feature and re-use it. Importantly, this plug-in system adds minimal overhead. Our commonly used features required fewer than 300 lines of Java code; we expect other feature plug-ins would require similar effort.

The feature table creation step enables users to use arbitrary kinds of sensors, including even complex sensors such as audio or video. A user simply needs to write plug-ins to extract the desired features from these sensors' data. Two of our contextual inquiry participants reported using automated feature reduction, so Gimlets also supported automated feature reduction via the underlying Weka tools.

Fourth, *after our first probe study* (with Group B, described below), we improved this visualization component to be more interactive for time-series sensor data as part of Gimlets 2.0. To do so, we added a time-window manipulation function to the graphs: When users drag a range of the time frame in the mini graph with the mouse, they can zoom in on that range of sensor data in the

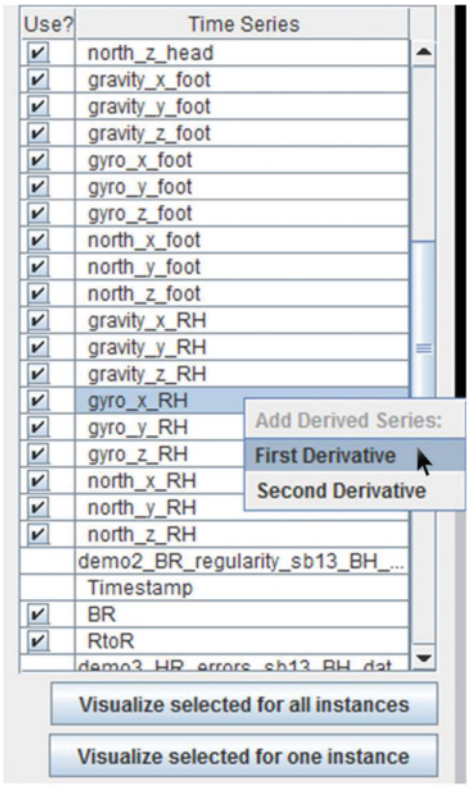


Fig. 5. Derived series creation.

main graph (Figure 7). When users hover the mouse over data plots on the graph, the value of the data point pops up (Figure 7, the yellow circle on the main graph).

In addition, two mixed-initiative features enable Gimlets to allow users to transform the format of raw sensor data (e.g., continuous time-series to discrete quantities), inspect data points in a similar pattern, and automate data batch processes (described in greater detail in Section 8.2).

Finally, we designed Gimlets to connect users to real-world contexts during data collection. The main graph displays information about the period of each session and annotated event. For example, Figure 3 shows driver activities annotated by computers or humans marked with red lines and white circles (upper main graph). When a user clicks on a circle, a panel with detailed information pops up around it. The red lines indicate when events of interest occur. Next, we embedded a multimedia player based on JavaFX (Figure 3, center pane). If videos are collected during data collection, then showing them synced with the sensor data (green vertical bar in Figure 7; vertical bar in each main graph in Figure 3) provides necessary real-world context for understanding the data.

Figure 8 shows a summary of the features that we implemented in the Gimlets system.

4.3 Model Evaluation and History Tree

We designed Gimlets to accommodate users' demands, which have been commonly highlighted in our contextual inquiries and in others' work with generic ML tools (e.g., Patel (2013)),that is, the iteration processes consisting of dashed arrows in Figure 2. Following the user-centric design process discussed in Amershi et al. (2014), we implemented end-user interaction based on

Fig. 6. Easier creation of feature tables for time series in Gimlets.

mouse input instead of custom scripts. However, Gimlets does not yet include interactive visualizations that could help users evaluate results (e.g., interactive visualizations demonstrated in ModelTracker (Amershi et al. 2015)), because our participants did not prioritize these iteration processes, compared to the domain-specific iteration processes related to sensor data and visual scan (round dot arrows in Figure 2).

In Gimlets, users can build, evaluate, and compare multiple models without writing custom scripts to interact with APIs. In our contextual inquiries, participants said that they would run through a range of models to see which would yield the best results and then dig into a particular classifier or test a more detailed configuration of it. Thus, we designed Gimlets to support this need.

For example, Gimlets' Build Models panel allows users to choose and apply any set of machine-learning classifiers, configure relevant parameters, and execute training processes (Figure 9(a)). If users select a trained model in the menu, then Gimlets displays information about evaluation results including accuracy, kappa, confusion matrix, and so on. In the Explore Results panel, users can view more detailed information such as how each instance is predicted or whether the prediction is correct (Figure 9(b)). For example, cells of the instances that are correctly classified are in blue and cells of the instances incorrectly classified are in red (Figure 9(b), Predicted column). The Compare Models panel allows users to track how and where performance deviates between baseline model and competing model (Figure 9(c)).

In both our contextual inquiries and Patel's studies, participants mentioned difficulties keeping their models and results organized. To aid in organization, we provided the History Tree panel (Figure 10), where users can view all current datasets, feature tables, and trained models, their classification results on the data and how they relate to each other. At any point in the process,

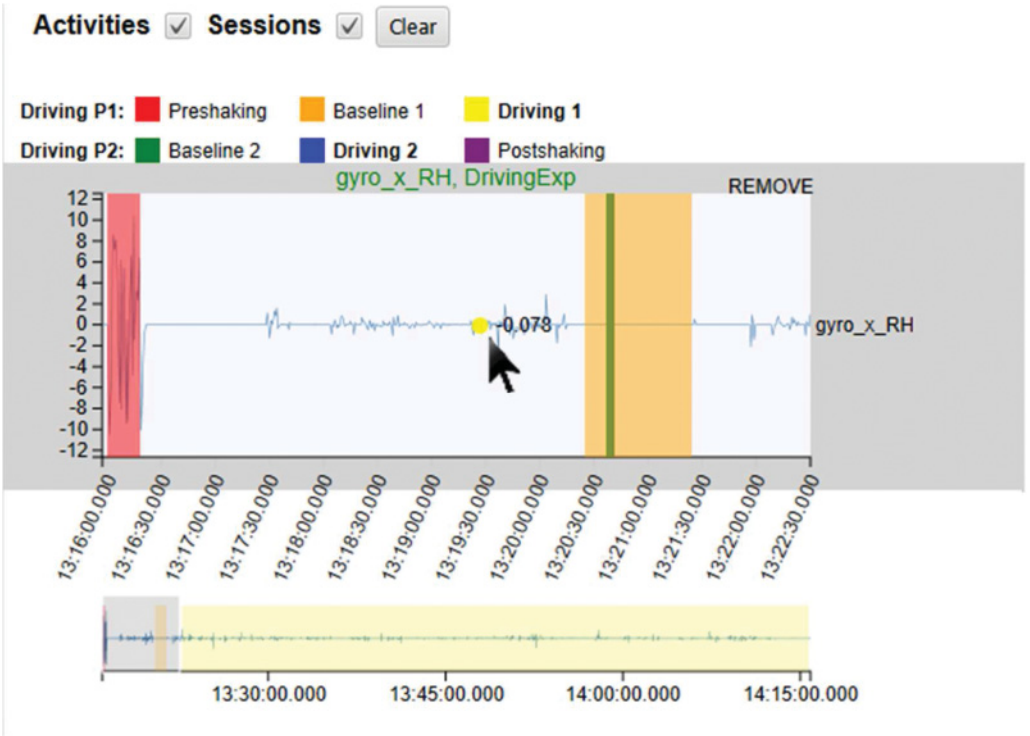


Fig. 7. Interactive visualization components in Load and Preprocessing Data panel.

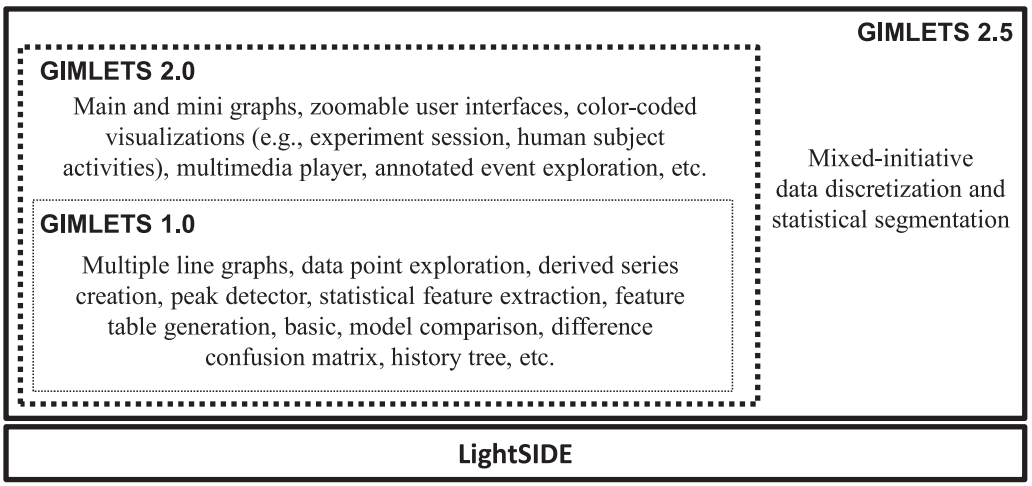
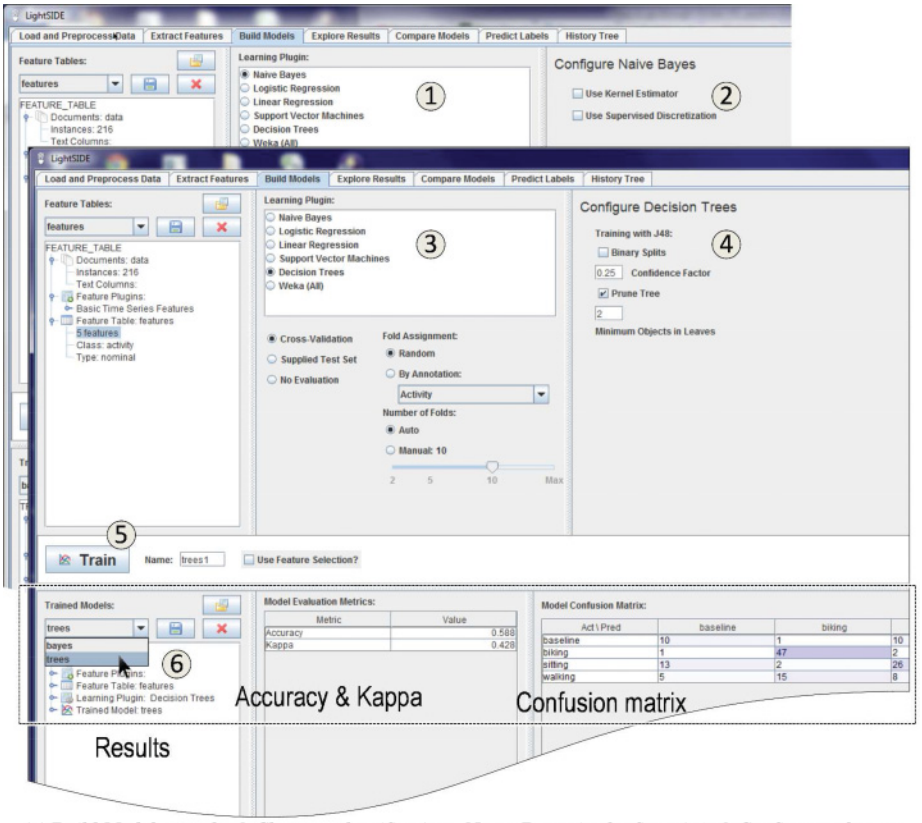


Fig. 8. Interactive visual-analytic and ML features in Gimlets 1.0, 2.0, and 2.5.

the user can see information about every step that led to the current model. For example, if a user has a set of raw data, and creates two different feature tables from it, the History Tree will display information about the dataset and both feature tables, with lines connecting the dataset to each feature table. Feature tables display information about what types of features were included,



(a) Build Models panel - ① Choose a classifier (e.g., Naïve Bayes in the figure) ➡ ② Configure relevant parameters ➡ ③ Choose another classifier (e.g., Decision trees in the figure) ➡ ④ Configure relevant parameters ➡ ⑤ Build models by clicking the Train button ➡ ⑥ Select a trained model in the menu to see the evaluation results

Fig. 9. Evaluation of performance of multiple models at a glance through (a) Build Models ➡ (b) Explore Results ➡ (c) Compare Models.

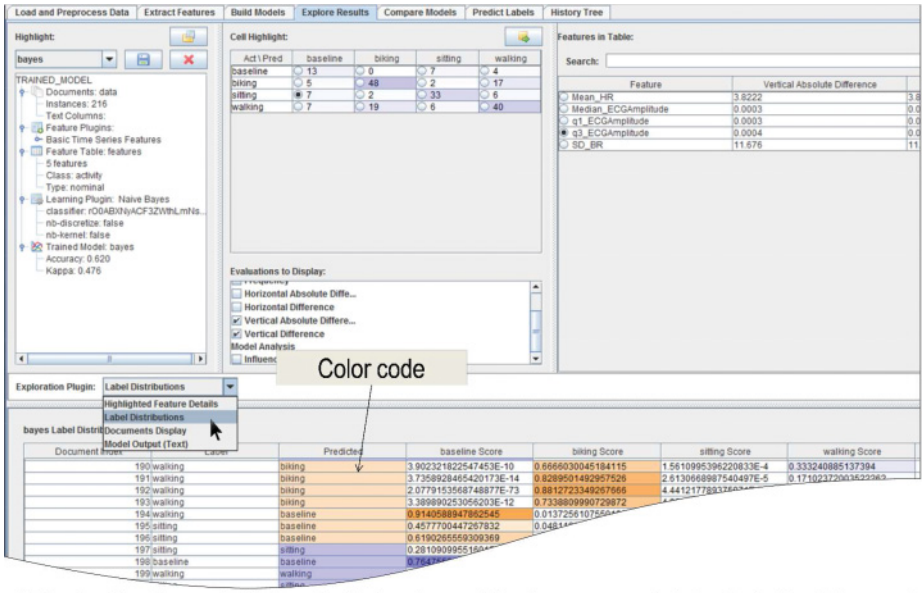
while trained models include information about the parameters they were trained with as well as the evaluation results of the models.

While similar features have been designed for non-sensor data (e.g., Gestalt's Hindsight), we added the History Tree feature to Gimlets to test the usability of hierarchical presentation of a history of the complex, iterative process for sensor data, because our participants echoed that sensor data classification required significant iteration in feature selection and model building.

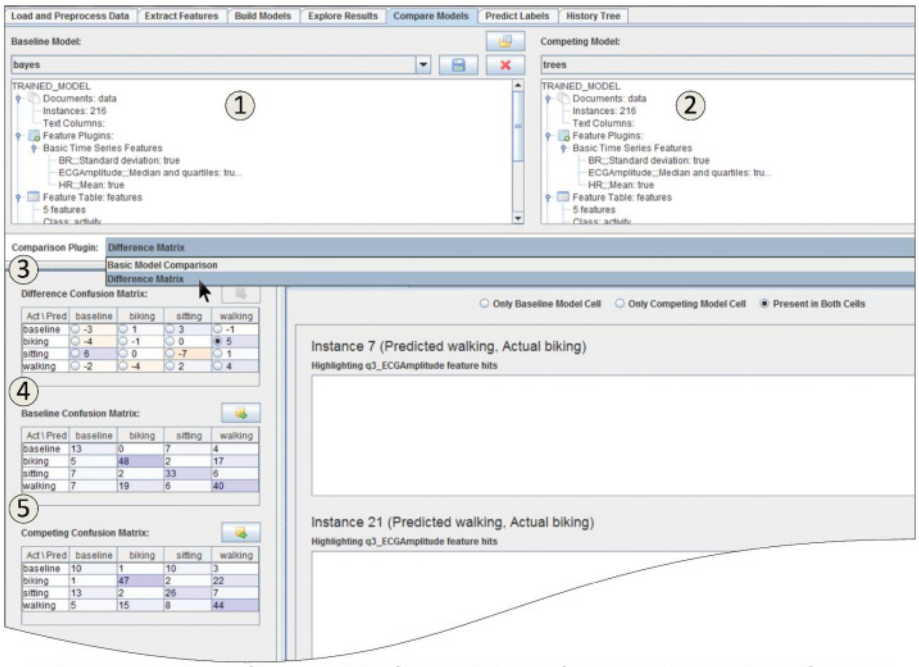
5 PROBE EVALUATION METHOD

To investigate which parts of Gimlets are the most useful to intermediate ML users, and to explore future design opportunities, we invited two groups of 10 participants to use Gimlets. Participants were computer scientists recruited from our university who have some prior experience with ML.

Participants provided demographic information and participated in a 10- to 15-minute instruction session. In this session, we reviewed the typical processes of ML tasks (Figure 2) and then introduced the overall ML pipeline in Gimlets (Figure 3), as well as its embedded visual analytic and ML features (Figures 5–7, 9, and 10).



(b) Explore Results panel - Color code displays how each instance was predicted and whether it is correct or not.



(c) Compare Models panel - ① Baseline Model, ② Competing Model, ③ Difference Confusion Matrix, ④ Baseline Confusion Matrix, ⑤ Competing Confusion Matrix.

Fig. 9. (Continued)

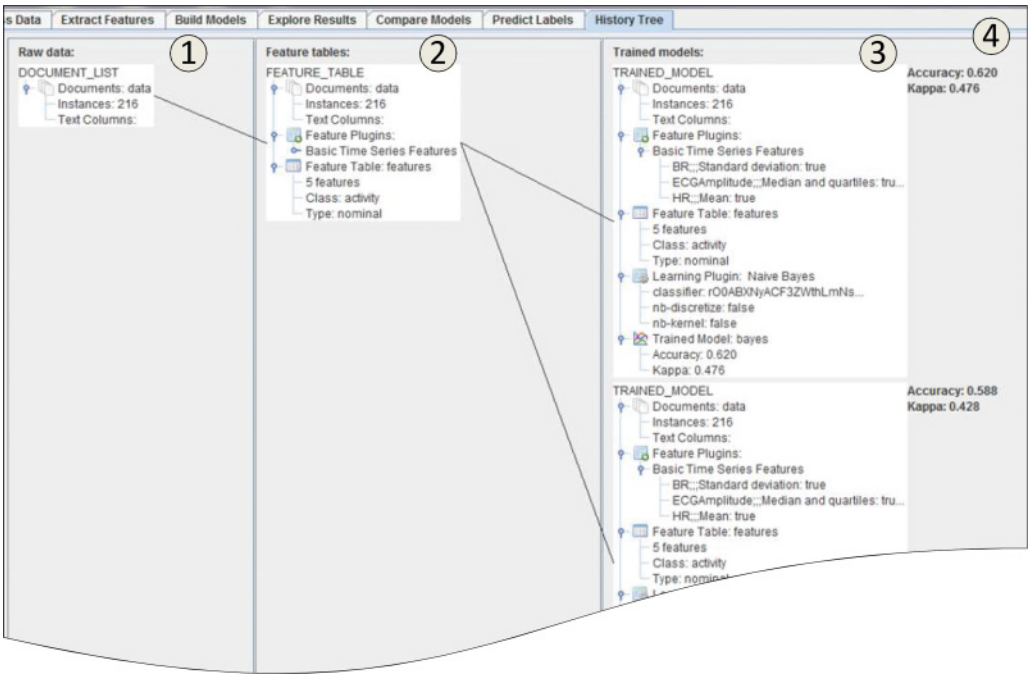


Fig. 10. History Tree panel in Gimlets ① Raw datasets, ② Feature tables, ③ Trained models, and ④ Evaluation results of trained models.

For the main tasks, which took between 40 and 60 minutes, Group B (Gimlets 1.0 users) classified physical activities such as walking, standing up, sitting down, lying down, bending, falling, and bicycling (using a stationary bike), which required them to explore sub-layouts across the ML pipeline. Group C (Gimlets 2.0 users) evaluated visual-analytic features embedded in the pipeline. One experimenter demonstrated data inspection tasks to participants, thus exposing them to all visual-analytics in the pipeline. After completing the given tasks, each group participated in an interview session and received $25.

Participants were asked to evaluate Gimlets with various big sensor datasets from four human-subject experiments with different classification purposes, thereby exploring the case study aspect. Participants also evaluated domain-specific features in Gimlets against corresponding features in their usual tools to explore the comparative evaluation aspect (mainly, in the second probe study). This complementary approach encouraged participants to evaluate the implemented features rather than common features in other tools and to discuss the features' domain-specific pros and cons.

Note that sensor data from body-worn sensors differs significantly from other datasets. If we were to start from raw data, then our end-users would complete numerous data parsing and time synchronization tasks across multidimensional sensors prior to the pre-processes we primarily explored in the study. To prevent these complications, we designed the studies to free our participants from the prerequisite tasks, which ranked low in our formative interviews.

In addition, to minimize the learning curve, we asked participants in Group C to compare Gimlets with their regular tools by grounding visual analytic features of Gimlets 2.0 in a single-level overview and detailed approach. This approach can extend to multi-focus interactions similarly to StackZooming (Javed et al 2010), which allows users to define and zoom in on multiple child strips

on the overview graph (i.e., mini graph in Gimlets) to compare time-series patterns between strips. However, we also freed our participants from such multi-focus interactivity in the current stage of this study, because our pre-interviews (concerning sensor data streams) showed considerable individual differences with regard to how our participants interactively manipulated voluminous datasets. We leave the extension to future work.

5.1 Probe 1—Classification Task (Group B)

In the first study, we probed the value of different features provided along with the visual layouts of Gimlets (Figure 3(a)). Ten participants (Group B with a few years of ML experience: $M = 4.3$, $SD = 1.8$; Figure 2, center) were asked to use Gimlets to develop a model for a dataset used in a previous study (Hong et al. 2013). This problem involved activity recognition, a classic ML task: Given sensor readings from a chest-belt type heart rate monitor, Android smartphone, and a radio tomographic imaging system, classify whether the user was standing, sitting, walking, or bicycling.

5.2 Probe 2—Visual Inspection Task (Group C)

The second study was focused on probing the usability of the interactive visual-analytic features provided in Gimlets 2.0. Another 10 participants (Group C; the right table in Figure 2) were shown how the updated features in Gimlets 2.0 could support a variety of visual-analytics tasks; these participants did not perform classification tasks. Most of our participants were intermediate ML users ($M = 4.4$, $SD = 1.4$), but 3 participants had over 10 years of experience each. This allowed us to assess whether more experienced users would see value in a tool like Gimlets.

We used datasets from two other previous studies (Figure 11). The first study (driver activity prediction) included video recordings of drivers and sensor data about their motion and physiological responses while driving (Kim et al. 2015). The other study involved early screening for autism and included video recordings and electrodermal activity data from a child and an examiner during a short interactive screening. Participants performed a series of three visual inspection tasks.

In the first task, participants were asked to inspect the session sequence in a driving experiment. Specifically, they checked the sequence of 20s sensor-shaking, 60s baseline measurement, and main driving sessions and then checked interactive visual analytic features between the main graphs and mini graphs and between the video player and the frame progress bar.

In the second task, participants were asked to inspect breathing rate (BR) regularity and erroneous heart rate (HR) in the same experiment (Figure 11(a)). Specifically, they checked differences in a BR data stream during baseline measurement sessions and main driving sessions and then scanned the time-windows that had missing or erroneous HR data, such as continuous errors and fluctuated errors during baseline measurement and driving sessions, respectively.

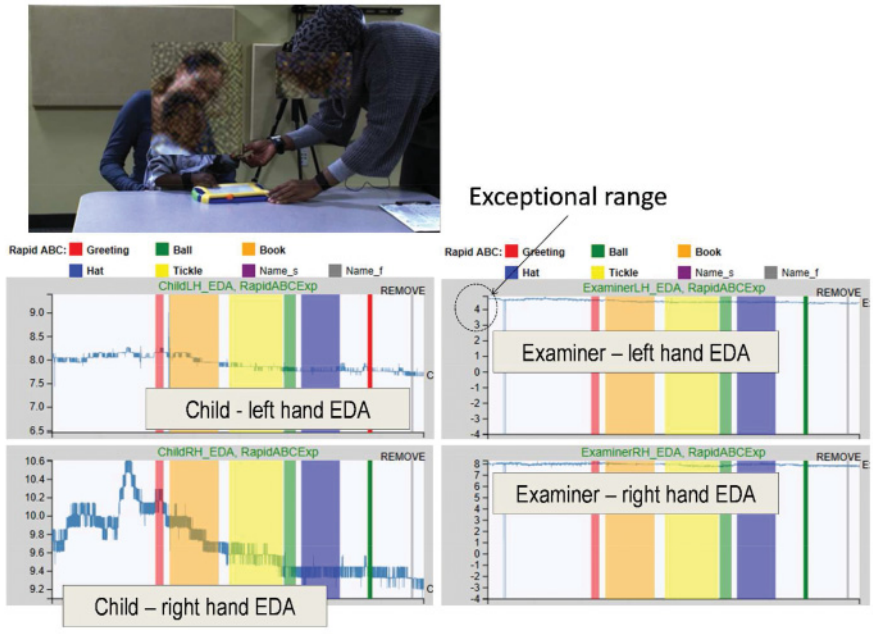
In the third task, participants were asked to inspect time-series data in an autism screener experiment (Figure 11b). They checked the sequence of pre-sensor reset sessions (i.e., pressing buttons on electro dermal activity, EDA, sensors), the series of main sessions corresponding annotations, and the post-sensor reset sessions. Participants also checked whether human annotation was properly completed (e.g., mismatched session time annotations; switched and lagged activity annotation) and if the human subject had skewed data in an EDA sensor data stream (which is difficult to determine errors or salient features at a glance).

As probe studies, we limited our evaluation based on a qualitative interview and a Likert-scale rating, without including user interaction logs or screen recordings.

6 PROBE RESULTS

After interviewing with each set of probe participants, we reviewed the interview recordings to identify key themes and insights into the features we implemented.

(a) Checking breathing rate (BR) regularity, erroneous heart rate (HR) in a driving activity prediction study



(b) Checking electro dermal activity (EDA) fluctuation in an autism screener

Fig. 11. Visual inspection tasks using Gimlets 2.0.

6.1 Group B: Value of Gimlets 1.0 Features

Participants in the first probe study, Group B, responded positively to three of Gimlets's main capabilities: easy feature table creation, derived series, and the history tree. Participants found loading data and creating feature tables in Gimlets easy and intuitive, especially the time-savings from being able to add built-in features like mean, standard deviation, and derivatives; most also mentioned a need to add more features via plug-ins.

B7 mentioned that, though some of these transformations would not be difficult to compute, being able to compute them with two clicks instead of writing code means "*you don't have to think about it… you are releasing that mental load.*" Users requested many transformations, including Fourier and wavelet transforms, and features like mean and standard deviation over sliding windows.

Six of our 10 participants in Group B expressed enthusiasm about interactive feature generation. We had included a "peak counter" feature, where users could select a y-value to count as a "peak" by clicking on the respective point in the graph. B5, B7, and B10 mentioned that interactive feature generation would need visual feedback to be useful, so users could try to specify a feature and then see where that feature appeared across many instances. In addition, B6 mentioned that interactive feature generation could lead to overfitting. Regardless, for some use cases, we believe that features could be specified interactively to save users time and frustration. As B9 said, "to go back and forth between toolkit and code is very frustrating," and interactive feature specification could help with this issue.

Nine participants in Group B appreciated the History Tree as well. They noted that the ability to compare models by accuracy within the history tree was crucial. Most said they would use it immediately, to make better models, but B5 said it would be helpful when he came back to his data a month later and needed to remember the steps he had taken. Seven of our participants liked the feature that allowed the comparing of models for statistical significance. B5 explained that he could use this to identify which models had the best performance (with no statistical difference) and then select the most intelligible of these models.

Our participants also expressed some frustration with using Gimlets for building a model. In particular, they wanted a more interactive data visualization to help them better understand their data, identify errors in the data, and generate hypotheses about useful features. In addition, they wanted the ability see available videos and class values with their time-series data to better understand the data.

6.2 Group C: Value of Interactive Visual Analytics in Gimlets 2.0

To address these issues, we iterated on the design of Gimlets and incorporated an interactive visualization system and a video playback panel to help with the unintelligibility of time-series data. This updated system (Gimlets 2.0 described earlier) was given to Group C.

All participants in Group C valued both of the new features. They consistently said the visual analytic functions enabled them to better understand what was occurring during data collection and supported visual inspection tasks for identifying erroneous data in physiological sensor data or issues with time synchronization of the sensor data. All of our participants who worked regularly with sensor data found this functionality quite appealing.

First, nine participants in Group C were quite impressed with the richer *interactive* visualization capabilities (e.g., zooming in on the main graph, popping up relevant sensor or annotation information by hovering the mouse point over the main graph) compared to their current tools. They were surprised to see them in an ML context, because most expected to use separate tools for visual inspection and ML.

C2 stated that Gimlets allowed end-users to *"actually position them [the sliders to zoom in or out main and mini graphs] properly based on the video and then just follow."* Therefore, he added, *"Being able to do that is really [useful], especially as one changes [the slider in the video player] as the [all sliders for main and mini graphs also] change direction… I can get a sense of data across different sensors."*

As C2 pointed out, the advantages of Gimlets' visualization system were to allow end-users to "*visually see any deviations from some sort of a normal sensory reading*" and to "*look at information from multiple sensors aligned together."* C6 also echoed that Gimlets helped people to *"visually inspect each of sources of data*" and *"compare it to each other where you're able to line up moments in time so you can see simultaneously what all the sensors are doing."*

In particular, C8 appreciated that Gimlets was superior to his current tool, Weka, in many aspects: "*[Regarding] Raw data visualization. In Weka, I still can't tell you what all these graphs mean because it doesn't necessarily… I can sit there and stare at them but normally they just tell you—this is how it relates to the class that you're identifying... This [Gimlets] is a bit more tangible but then again it assumes the data I have is a video... [Regarding] Brainstorming new features to include. I would say it [Gimlets] is better than Weka but Weka sets a very low bar in that area. So, this could potentially be more useful… [Regarding the detection of] Unusual human subject. I would say [Gimlets] is a lot better because Weka likes to sum up all the instances but it doesn't like to tell you about a particular instance. … [Regarding] Get a sense of how to process that data. I'm going to say that Weka does not help me. It just gives me tools for finding, it just lets me throw as many machine learning algorithms at it to see what sticks.…"*

All participants in Group C expressed appreciation for Gimlets' ability to support user control for voluminous time-series sensor data, whereas only two Matlab users and one Python user, who already had their own code for raw data visualization customized with their ML pipeline, were concerned about the learning curve of our system. Still, like the other seven participants who relied on existing tools without their own custom code, two of the three Group C participants preferred Gimlets over their own code. Participant remarks about a potential learning curve primarily arose from their competence or familiarity with their own code or tool usage, rather than evidential observation about potential complexities of Gimlets control. For example, C1 stated, "*I know I use Matlab because I know I can control – like it's not easy but I can do what I want, whereas this [Gimlets] looks more intuitive and helpful, but if I can't do something that I want it to do then that's one of the things that I find frustrating about a tool like this.… In Matlab, I'm like 'Okay, well it will take a fair amount of code but I can do it.' Something like this I'm not sure—well I guess this is part of the learning curve.*"

Additionally, our participants appreciated that the new visual analytic features, which we included in Gimlets 2.0 to help end-users comprehend real-world contexts, improved the intelligibility of time-series sensor data, as revealed in the following participants' comments:

- C2: *"One thing that I particularly like is this validation of your labels where you can go through the data and see it labeled with these colors and say 'this doesn't make any sense or maybe this label needs to be split.' Maybe within that particular label, we're seeing lots of differences that are clear."*
- C6: *"I think it is useful to be able to see the video because if you're looking at some weird fluctuation in one of the sensors, you can look at the video and try to see why. So it's got some kind of explanatory power."*
- C8: *"The advantage there is that you can actually see the video tied to the actual data. I like that you can actually get a sense of looking at the data, seeing something that's wrong and then being able to verify it quickly in the video… it's no longer just data, it's data tied to a video and the fact that you can go between it. That's important because, [for] example [,]*

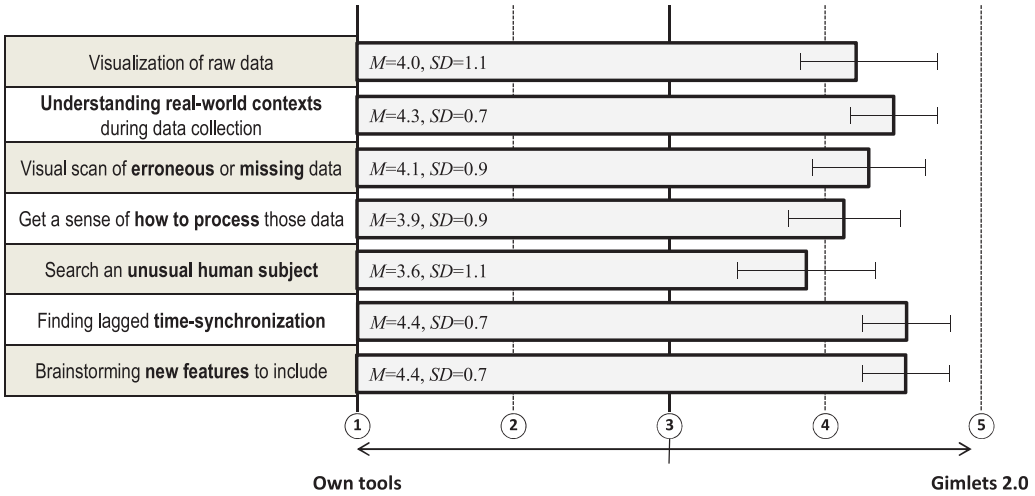


Fig. 12. Evaluation results of Gimlets 2.0 compared to participants' own tools.

> *understanding [changes of] one's skin temperature, seeing a chart of it doesn't mean anything until you understand the context of it. For example, those 200 heartbeats per minute could have been somebody who had just gotten into an accident. I guess if I just saw a spreadsheet with that type of information, it would be hard for me to immediately to tell you that this is wrong. I see it on the video and I can say clearly that this person doesn't have 200 bpm or she has a huge problem that we should let her know about."*

C8 appreciated that Gimlets' visual analytic functions were distinct from his current tool, Matlab: *"The video is a definite improvement. Like that's one thing Matlab does not really do well. Visual data. The errors I feel like are fine. Get a sense of how to process...yeah, I think that some of that is helpful. Like with labeling and things is nice here that Matlab doesn't do particularly well... I like the labeling part and video stream. And yeah, definitely compared to Matlab, that's a big improvement."*

Finally, participants rated how the visual analytic aspects in Gimlets compared to the tools they regularly used for performing machine-learning tasks, using a five-point Likert-scale (1: strongly prefer own tools, 3: neutral, 5: strongly prefer Gimlets). The tasks, ordered by ratings, are as follows: identify new sensor features to include in classification task; identify issues with time-synchronization of data; understand real-world context of data collection; identify erroneous or missing data; visualize raw data; understand how to identify and deal with erroneous and missing data; identify outliers in the subject population.

Figure 12 shows that, on average, for all tasks, participants preferred Gimlets to their tools. These results are supported by participant feedback as well. For example, C7 stated, "*For time series this one [Gimlets] would really show me where it is. The processing part is better because there aren't a lot of good time series tools. I can't give you another example that does this. I may even wanna use this [Gimlets] for my own research. This is the first time I'm seeing this... The interface is very clear that everything is there. I'm no longer concerned about raw data. I can see if I have everything.*"

7 DISCUSSION

In addition to feedback on specific features, a few higher-level points emerged from our study.

7.1 Dual Purpose Task: Accuracy vs. Understanding

When we asked our participants to build a model, we assumed, based on Patel et al. (2008a), that understanding the data and models would be a critical part of the process. Indeed, for six participants in Group B it was. They added features judiciously, both to avoid overfitting and to better understand which features improved classification accuracy. B4 even suggested starting with 1-2 features and then adding features one by one, training and evaluating a model each time, to understand the effect of each feature. These participants also chose models that could be explained, like decision trees, so if a model were successful, its classifications would be easy to understand.

For the six participants, ML was primarily a tool to aid understanding. Developing one model to classify one dataset was not useful; they wanted to learn principles that would help them better classify and understand future data. Being able to explain their models to colleagues was considered important, too. B3, one of our intermediate ML users who works in learning science, explained that if he included too many features or selected a model carelessly, "*You're not going to get those results accepted because basically what you're saying is, we don't know what we're doing, so we found a machine learning algorithm that for this data and these attributes give me really good results, we just don't know why.*"

However, four participants in Group B simply tried to build the most accurate model possible. They mostly used a brute-force approach in both feature generation and model tuning. All four used all of the available features, trusting that either automated feature selection would remove unhelpful features or that the model would be able to weigh all of the features appropriately. Three of these also explored different model types and parameters in more depth than most other users, trying many different models to find the one with the highest accuracy. B2, an ML researcher with 7 years of ML experience in the Machine Learning department, summarized this approach: "*Since I do machine learning and focus on the algorithms, I usually just treat features as a black box and don't really care what they actually mean.*"

Because users will use both approaches, depending on their tasks, we recommend designing for both perspectives. For example, feature selection tools should allow users fine-grained control but also a "select all" button to more easily enable the brute force approach.

7.2 Mixed-Initiative Support Is Valuable

Users in Group B expressed a desire for a more proactive tool, to check for data errors, suggest best practices, suggest particular models given the characteristics of the data being classified, and to automatically, in the background, explore the value of potential features. B1 and B3 suggested that the tool scan the data to eliminate missing values or values that are always zero; B3 also suggested some simple distribution tests on the data to highlight anomalies. B6 suggested, "*The path of least resistance to making a model should have all of the best practices sort of included, standard.*" Similarly, others suggested automating a split of the data into validation, training, and test sets, following best practices, to avoid overfitting features to the training set. In addition, the tool could try many different models or features in the background and suggest those to help guide the user. Therefore, after the Group B study, we thought that future tools should focus on embedding automated support.

However, when shown more powerful visualizations in Gimlets 2.0, users in Group C emphasized a need for *in situ* action support. They wanted to process erroneous, missing, or skewed data; create new sensor features; and browse summary statistics anywhere, even on video scenes. Most of our participants expected that a future version of Gimlets would facilitate error search and removal (e.g., highlighting ranges where data integrity was uncertain) and enable users to manipulate timestamps across sensor data streams to address issues with synchronization

without manually editing and reloading the sensor CSV datafiles. C8 said, "*it seems like there's still a manual phase where you say I'm interested in this particular segment of data. I can see it but I don't know if the tool is actually going to align it for me. Maybe that's what I actually need.*" Also, it was expected that Gimlets should enable users to interactively combine and separate displayed sensor features to create new sensor features (e.g., drag a graph of breathing data and drop it onto body motion data to produce new driver state features).

Also, four participants suggested that our system should provide constant access to summary statistics for each person's data in the given experiment; for example, when viewing a short segment of heart rate data, they wanted to easily determine if that rate was abnormal. Further, two of these participants wanted real-time data to be dynamically overlaid on the video scene (e.g., displaying car speed on the driver video).

We see these results as pointing to a mixed-initiative interaction. Some parts of the model creation process would be initiated by the system, such as highlighting the maximum and minimum data values in each sensor stream and analyzing different feature sets and classifiers in the background. Other aspects would be performed by the user with access to advanced visual analytics support, such as confirming which data values were anomalous, data alignment, and generating and validating hypotheses about the value of particular sensor streams and features.

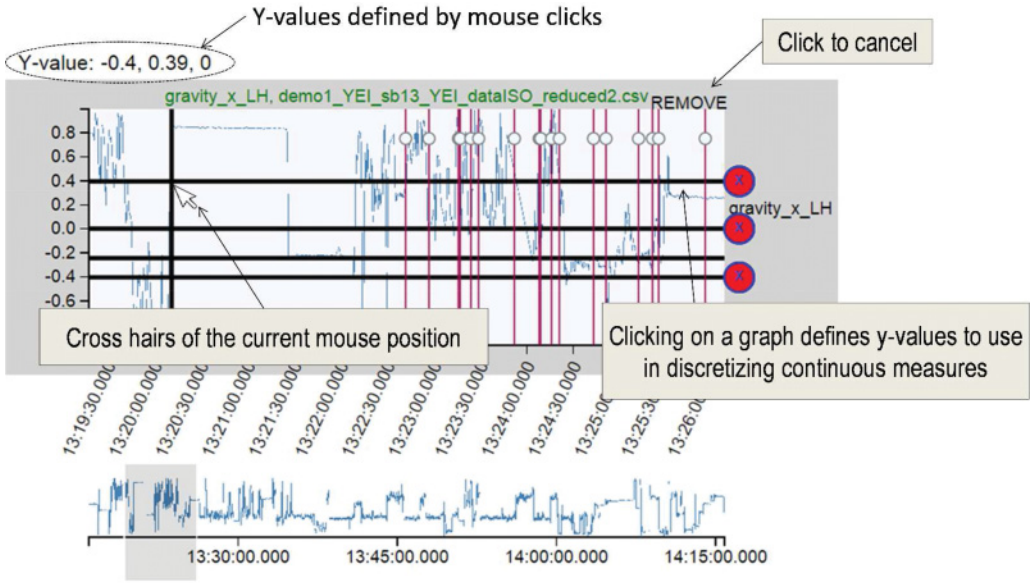
Given participant feedback, we implemented two new mixed-initiative features, discretization and segmentation, for Gimlets 2.5 (Figure 13). First, discretization allows users to transform time series of numbers into discrete quantities to better support evaluation. When users click on a graph, a set of *y*-values are defined and then discrete features are automatically generated from raw data (e.g., labeling `Low` for sensor data points which are smaller than -0.4, `Medium` for data points between -0.4 and 0, `High` for data points between 0 and 0.39, and `Very High` for data points that are larger than 0.39, Figure 13(a)). Second, segmentation allows users to easily search data points in a similar pattern and then automate batch processes for those data points (Figure 13(b)). If users drag a range of data (e.g., data which look erroneous or missed), then the feature automatically searches and highlights similar data points to help users process them *in situ* (e.g., remove them).

7.3 The Process Has Many Steps

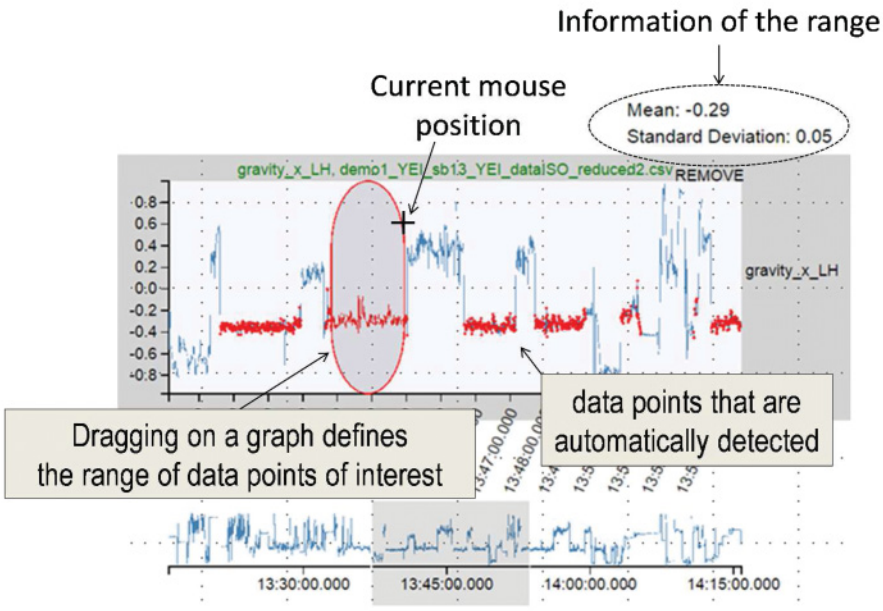
Participants in studies A and B suggested many steps in the pipeline besides what we explicitly supported in our tool. Some suggested principal component analysis or other methods of feature reduction, combining the results of multiple models, and automatically breaking the dataset into subsets for quicker iteration. This last suggestion echoes that of A2 and A3, who split their data into different subsets for different reasons. Patel et al. found this diversity in the process steps as well, which is why Gestalt offered the ability to add arbitrary data processing steps (Patel et al. 2010). From our results, we suggest that future tools need to explicitly guide less-experienced users through the many steps.

7.4 A New Sandbox for Feature Tables

Gimlets was designed as a probe to investigate the value of different functionality for classifying sensor data. However, we did ask participants if they would use Gimlets as is. In group B, five of our least experienced participants said they would; Group C, using Gimlets 2.0, were much more positive, with 9 of 10 saying they would. Reasons that participants would not use Gimlets mostly included adoption barriers and familiarity with existing tools. Three participants from Group C developed their own code in Matlab and Python and expressed some concerns about the learning curve for adopting Gimlets. As B5 said, using a tool like Gimlets would involve writing plug-ins, and "you really only feel like investing time into tools you love." B8 noted that he usually cleaned



(a)



(b)

Fig. 13. New features implemented for Gimlets 2.5 – (a) discretization and (b) segmentation.

his data in Matlab at the start of a project, so it is easier for him to continue using Matlab to generate features and build models. B1 noted an interesting cultural concern, however: "*From an ECE [Electrical and Computer Engineering] Ph.D. standpoint, it's expected that I develop my own algorithms.*" This further supports our notion that for novice users, tools like Gimlets will help them produce classification models without requiring many years of practice, but then as users become more expert, they may outgrow these tools.

Despite this, more experienced users (e.g., B7) found value in the simple steps, like easily creating derived series, because "*[the user does not] have to think about it*" and can focus on the more thought-intensive parts of the process. Providing support for plug-ins that allow re-usability across users means that experienced users can quickly manipulate their data and build models that explore many more possibilities than they currently try because of the friction involved in writing custom code. We expect that this re-use will save time and result in better models and systems, but this needs to be explored in future work.

B9 reported that "*people always prefer using a toolkit,*" and while this may not be true for everyone, B6 agreed: "*I like the idea of not having to write code every time I build a classifier.*" He mentioned that Gimlets was a "*slightly bigger sandbox [than Weka].*" While a bigger sandbox is more powerful, a smaller one is easier to use, so he preferred the smallest sandbox possible. Even more experienced users like B2 and B7 mentioned the possibility of using tools like Gimlets to automatically generate many derived time series and many features. These results show us that there is some value in a plug-in-based toolkit.

We received quite positive feedback about the interactive visualizations to support increased understanding of data and to make it easier to generate features, about the plug-in architecture that supports extensibility, and about the history tree to better track one's model creation process and compare the results of different feature sets and classifiers. We believe that future tools for intermediate ML users doing sensor data mining should focus on these pre-processing and post-processing steps and embed them within an ML pipeline to support the training of classifiers. Secondarily, future tools should focus on compatibility with existing tools for more experienced users (e.g., Weka and Matlab) with which ML users have written their own custom code. Further, future tools should support mixed-initiative interaction, allowing both users and the tool to explore or suggest data for cleaning, and features/classifiers that could be used. Finally, for less experienced users, these tools should provide a guided and structured process for creating classifiers.

8 CONCLUSION

Classification of sensor data remains a useful procedure that enables many important applications. However, this task remains difficult except for a small group of experts. Intermediate ML users still do not have adequate tools to help them build systems involving sensor data classification, due to the volume, multiple sources, and unintelligibility of sensor data. Through introductory inquiries and two prototype-based user studies, we evaluated the usefulness of multiple features and uncovered further recommendations for future interactive ML systems. We hope to continue by building a tool that helps intermediate users to build systems involving classification of sensor data, greatly expanding the number and quality of these useful systems.

REFERENCES

Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. 2011. Visualization of time-oriented data. *Springer Science & Business Media.*

Saleema Amershi, James Fogarty, and Daniel Weld. 2012. Regroup: Interactive machine learning for on-demand group creation in social networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12). ACM*, New York, NY, 21–30.

Saleema Amershi. 2012. Designing for effective end-user interaction with machine learning. *Ph.D. Dissertation. University of Washington, Seattle, WA*.

Saleema Amershi, Maya Cakmak, W. Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Mag. 35*, 4, 105120.

Saleema Amershi, Max Chickering, Steven M. Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. 2015. ModelTracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'15)*, 337–346.

Ling Bao and Stephen S. Intille. 2004. Activity recognition from user-annotated acceleration data. In *Proceedings of the 2nd International Conference on Pervasive Computing*, LNCS 3001, 1–17.

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, New York, NY.

Tanzeem Choudhury, Gaetano Borriello, Sunny Consolvo, Dirk Haehnel, Beverly Harrison, Bruce Hemingway, Jeffrey Hightower, Predrag "Pedja" Klasnja, Karl Koscher, Anthony LaMarca, James A. Landay, Louis LeGrand, Jonathan Lester, Ali Rahimi, Adam Rea, and Danny Wyatt. 2008. The mobile sensing platform: An embedded activity recognition system. *IEEE Perv. Comput.* 7, 2 (April 2008), 32–41.

Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. a CAPpella: Programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04). ACM*, New York, NY, 33–40.

Richard O. Duda and Peter E. Hart. 1973. *Pattern Classification and Scene Analysis*. Vol. 3. New York: Wiley, 1973.

Jerry Alan Fails and Dan R. Olsen, Jr. 2003. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI'03).* ACM, New York, NY, 39–45.

James Fogarty, Desney Tan, Ashish Kzpoor, and Simon Winder. 2008. CueFlik: Interactive concept learning in image search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08).* ACM, New York, NY, 29–38.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.* 11, 1 (November 2009), 10–18.

Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'07).* ACM, New York, NY, 145–154.

Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST'06).* ACM, New York, NY, 299–308.

Douglas M. Hawkins. 2004. The problem of overfitting. *J. Chem. Inf. Comput. Sci.* 44, 1, 1–12.

Jin-Hyuk Hong, Julian Ramos, Choonsung Shin, and Anind K. Dey. 2013. An activity recognition system for ambient assisted living environments. In *Evaluating AAL Systems Through Competitive Benchmarking (EvAAL'12), Communications in Computer and Information Science*, S. Chessa and S. Knauth (Eds.). Vol. 362. Springer, Berlin, Heidelberg.

Jin-Hyuk Hong, Julian Ramos, and Anind K. Dey. 2012. Understanding physiological responses to stressors during physical activity. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp'12).* ACM, New York, NY, 270–279.

Deng-Yuan Huang, Wu-Chih Hu, and Sung-Hsiang Chang. 2009. Vision-based hand gesture recognition using PCA + Gabor filters and SVM. In *Proceedings of the 2009 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'09).* IEEE Computer Society, Washington, DC, 1–4.

Jun Kato, Sean McDirmid, and Xiang Cao. 2012. DejaVu: Integrated support for developing interactive camera-based programs. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST'12). ACM*, New York, NY, 189–196.

Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. 2008. Visual analytics: Definition, process, and challenges. In *Information Visualization*, Andreas Kerren, John T. Stasko, Jean-Daniel Fekete, and Chris North (Eds.). Lecture Notes in Computer Science, Vol. 4950. Springer-Verlag, Berlin, 154–175.

SeungJun Kim, Jaemin Chun, and Anind K. Dey. 2015. Sensors know when to interrupt you in the car: Detecting driver interruptibility through monitoring of peripheral interactions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'15*). ACM, New York, NY, 487–496.

Khee Poh Lam, Michael Höynck, Bing Dong, Burton Andrews, Yun-Shang Chiou, Rui Zhang, Diego Benitez, and Joonho Choi. 2009. Occupancy detection through an extensive environmental sensor network in an open-plan office building. In *Proceedings of the International Building Performance Simulation Association Conference (IBPSA'09)* 1452–1459.

Elijah Mayfield and Carolyn Penstein Rosé. 2013. *LightSIDE: Open Source Machine Learning for Text*. In M. D. Shermis and J. C. Burstein (Eds.), Handbook of automated essay evaluation: Current application and new directions. Psychology Press, New York, NY, 124–135, 2013.

Dan Maynes-Aminzade, Terry Winograd, and Takeo Igarashi. 2007. Eyepatch: Prototyping camera-based interaction through examples. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST'07)*. ACM, New York, NY, 33–42.

Kayur Patel. 2013. *Lowering the Barrier to Applying Machine Learning*. Ph.D. Dissertation. Computer Science and Engineering, University of Washington, Seattle, WA.

Kayur Patel, Steven M. Drucker, James Fogarty, Ashish Kapoor, and Desney S. Tan. 2011. Using multiple models to understand data. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, Toby Walsh (Ed.), Volume Two. AAAI Press 1723–1728.

Kayur Patel, Naomi Bancroft, Steven M. Drucker, James Fogarty, Andrew J. Ko, and James Landay. 2010. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology (UIST'10)*. ACM, New York, NY, 37–46.

Kayur Patel, James Fogarty, James A. Landay, and Beverly Harrison. 2008a. Examining difficulties software developers encounter in the adoption of statistical machine learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence* (AAAI'08), Anthony Cohn (Ed.), Vol. 3. AAAI Press 1563–1566.

Kayur Patel, James Fogarty, James A. Landay, and Beverly Harrison. 2008b. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*. ACM, New York, NY, 667–676.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (November 2011), 2825–2830.

Jian Zhao, Fanny Chevalier, Emmanuel Pietriga, and Ravin Balakrishnan. 2011a. Exploratory analysis of time-series with chronolenses. *IEEE Trans. Vis. Comput. Graph.* 17, 12 (December 2011), 2422–2431.

Jian Zhao, Fanny Chevalier, and Ravin Balakrishnan. 2011b. KronoMiner: Using multi-foci navigation for the visual exploration of time-series data. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'11)*. ACM, New York, NY, 1737–1746.

Received July 2015; revised March 2016; accepted May 2016