Vizdom: Interactive Analytics through Pen and Touch

Andrew Crotty   Alex Galakatos   Emanuel Zgraggen
Carsten Binnig   Tim Kraska
Department of Computer Science, Brown University
{firstname_lastname}@brown.edu

ABSTRACT

Machine learning (ML) and advanced statistics are important tools for drawing insights from large datasets. However, these techniques often require human intervention to steer computation towards meaningful results. In this demo, we present VIZDOM, a new system for interactive analytics through pen and touch. VIZDOM's frontend allows users to visually compose complex workflows of ML and statistics operators on an interactive whiteboard, and the backend leverages recent advances in workflow compilation techniques to run these computations at interactive speeds. Additionally, we are exploring approximation techniques for quickly visualizing partial results that incrementally refine over time. This demo will show VIZDOM's capabilities by allowing users to interactively build complex analytics workflows using real-world datasets.

1. INTRODUCTION

Visualizations are one of the most important tools for exploring, understanding, and conveying facts about data. However, the rapidly increasing volume of data often exceeds our capabilities to digest and interpret it, even with sophisticated visualizations. Traditional OLAP-style reporting can offer high-level summaries about large datasets but cannot reveal more meaningful insights. On the other hand, complex analytics tasks, such as machine learning (ML) and advanced statistics, can help to uncover hidden signals.

Unfortunately, these techniques are not magical tools that can miraculously produce incredible insights on their own; instead, they must be guided by the user to unfold their full potential. For example, a recent study [5] that used Twitter data to predict national unemployment rates demonstrates the importance of choosing good features, since the authors discovered that a knowledge-driven approach using carefully selected phrases (e.g., "I need a job") outperformed a model trained over the complete bag of words. However, the process of finding these features is often the result of iterative trial-and-error, where a domain expert tests different subsets of features until finding those that work best.

Proceedings of the VLDB Endowment, Vol. 8, No. 12
Copyright 2015 VLDB Endowment 2150-8097/15/08.

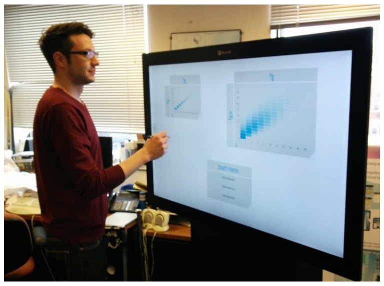

Figure 1: VIZDOM on an Interactive Whiteboard

Furthermore, users often need to carefully tune algorithm-specific hyperparameters in order to obtain good results. K-means clustering, for instance, requires the user to pick an appropriate number of centroids, while supervised classification tasks require the selection of an appropriate kernel. Additionally, these ML algorithms rarely work in isolation, and they are usually combined into complex workflows involving preprocessing, training, and evaluation.

While human input is clearly essential for deriving useful insights using ML, existing frameworks are not designed for iteratively refining complex workflows in an interactive way. Most existing libraries (e.g., Weka [7], Mahout [2], MLlib [11], MADlib [8]) focus on executing a single algorithm at a time with fixed parameters. These libraries represent a black-box approach where users pick an algorithm and wait potentially hours for a result, at which point they frequently need to tune parameters or try a different algorithm. MLbase [9] tries to improve the model refinement process by automatically selecting tuned algorithms but lacks the human element that is crucial for extracting actionable knowledge. Other tools like RapidMiner [4] allow users to visually create ML workflows but are still far from delivering results within the interactivity threshold (around 500ms [10]).

In this demo, we present VIZDOM (Figure 1), a new system for interactive analytics through pen and touch. Our frontend was designed for visual interfaces that allow users to interactively explore data using sophisticated visualizations, and our backend leverages a high-performance distributed analytics framework to achieve interactive speeds. Furthermore, we are investigating new approximation techniques for visualizing partial results. This demo will allow users to visually compose complex analytics workflows using several real-world datasets.

2. VIZDOM ARCHITECTURE

We envision a complete paradigm shift in how data scientists conduct exploratory analytics. Rather than several back-and-forth interactions between data scientists and domain experts, we believe that the two can work together using a large interactive whiteboard to visualize, transform, and analyze data on the spot, allowing them to quickly arrive at an initial solution that can be further refined offline. This section describes the architecture of VIZDOM, which has two parts: (1) a frontend built on PanoramicData, a visual interface for data exploration; and (2) a backend built on Tupleware, a high-performance distributed analytics framework. Additionally, we outline some approximation techniques that we are investigating to improve the interactivity of complex analytics tasks.

2.1 Frontend

Most users interact with data through traditional query languages like SQL and pre-canned reports that do not allow for free-form data exploration. Instead, we believe users need a visual interface in which they can quickly test out a wide variety of hypotheses with minimal effort as they interactively refine these hypotheses over time.

PanoramicData [13] is a visual frontend to SQL that allows users to rapidly search through datasets using visual queries constructed by pen and touch manipulation. For VIZDOM, we extended PanoramicData's functionality to include sophisticated ML and statistics operators.

Figure 2 shows the primary VIZDOM interface, which is a blank canvas onto which users can drag either individual attributes or predefined operators. Pre-loaded datasets are listed along the bottom, while the attributes for the selected dataset and available operators are displayed along the left-hand side. By default, dragging an attribute onto the canvas creates a histogram showing the distribution of the data, as shown by the visualization of the age attribute in the figure. Users can also modify these visualizations to plot different attributes against each other (e.g., height vs. weight), producing a two-dimensional histogram. On the other hand, dragging an operator onto the canvas provides a template for a task (e.g., logistic regression) that users can parameterize with attributes from the dataset.

Moreover, users can link together operators to easily create different views over the data, since visualizations also act as data filters. For example, a user could select a subset of the data in Figure 2 by performing a lasso gesture with the pen. Modifying an individual visualization (e.g., selecting a subset based on age) triggers an immediate re-execution of all linked downstream visualizations, with intermediate results cached for reuse in future dependent visualizations. This feature allows users to interactively refine their hypotheses based on the latest results and steer the computation to find interesting patterns in the data.

2.2 Backend

The requirement to provide interactive speeds for complex analytics tasks poses entirely new challenges for the backend. As shown by a recent study [10], even slight delays during visual interaction substantially reduce the willingness of the user to explore the data and test new hypotheses. We therefore argue that a visual data analysis framework must avoid visualization delays at all costs. With this goal in mind,

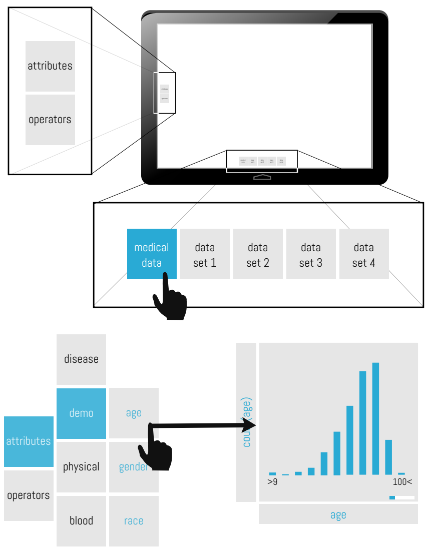

Figure 2: VIZDOM **Interface**

we built VIZDOM's backend on Tupleware [6], a new high-performance distributed analytics system that addresses the interactivity challenge through three key ideas.

(1) Small Clusters: Current systems for complex analytics (e.g., Hadoop [1], Spark [12]) are designed for large cloud deployments running on cheap commodity hardware. While these frameworks are well-suited for large-scale batch processing, they are a notoriously bad fit for interactive visualizations where low latency is paramount. For example, anecdotal experience with Spark has shown that job scheduling and deployment takes several hundred milliseconds, which already exceeds the interactivity threshold without even performing any of the actual computation. Instead, we believe that interactive analytics tasks should be run on small clusters of high-end hardware (e.g., abundant main memory, high-speed networks), and Tupleware was specifically designed for this type of infrastructure.

(2) Low-Level Optimizations: Users typically express complex analytics tasks as a workflow of operators that define the different steps (e.g., cleaning, filtering, training) of an ML pipeline. Many of these steps are heavily CPU-bound, and the backend must optimize specifically for this computation bottleneck to guarantee interactive speeds. Tupleware addresses this problem by compiling workflows directly into self-contained distributed programs, improving performance by eliminating common sources of overhead (e.g., external function calls, polymorphic iterators) and applying traditional compiler techniques (e.g., inline expansion, SIMD vectorization). As part of the compilation process, Tupleware dynamically generates all of the necessary control flow, synchronization, and communication code.

(3) Shared State: Tupleware natively incorporates the notion of globally distributed shared state, which is a key ingredient of many ML algorithms. Other attempts to support distributed shared state impose substantial restrictions on how and when programs can interact with global variables. Not only do Tupleware's efficient shared state mechanisms improve the performance of ML tasks, they also allow the system to stream results to the visual frontend at safe, well-defined, and algorithm-specific points.

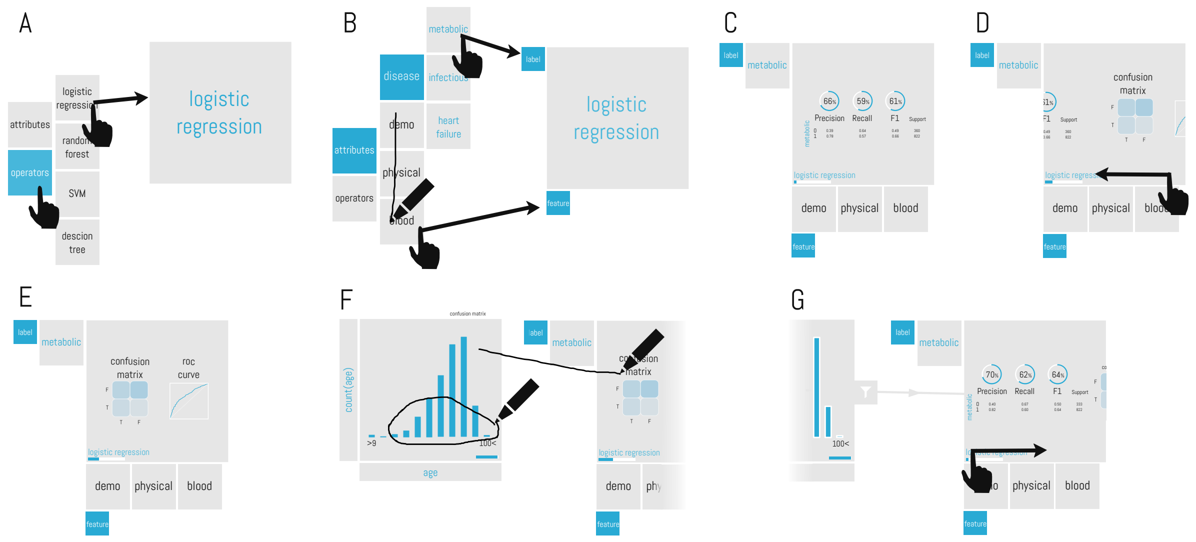

Figure 3: Story Board I

2.3 Approximation Techniques

Even with the fastest backend, the time necessary to process a sufficiently large dataset can still exceed the interactivity threshold. To speed up these tasks, we are investigating several techniques for quickly providing approximate results that refine over time, indicated by the progress/error bars accompanying each visualization.

(1) Dynamic Sample Resizing: VIZDOM implements a form of dynamic sample resizing, which begins by training and evaluating a model using only a small subset of the dataset to obtain quick initial results. The sample size can then be incrementally increased to yield a more accurate result until arriving at a final solution. This idea is similar to a technique implemented in Columbus [14], which uses a small sample to initialize a model in order to converge faster. However, instead of only improving the convergence rate, we also wish to leverage sampling to provide quick approximate results to the visual frontend.

(2) Incremental Cross-Validation: By default, VIZDOM uses 10-fold cross-validation as the default quality evaluation metric. However, instead of forcing the user to wait until all 10 folds complete, we can incrementally stream partial results to the visual frontend after each fold, with result quality improving over time.

(3) Hyperparameter Adjustment: Many ML algorithms have tunable hyperparameters that significantly impact the quality of the resulting model, but there is a tradeoff between model quality and training time. For example, a random forest classifier is an ensemble learning method that builds several independent decision trees and combines the results using a voting scheme. In this case, VIZDOM could start with a small number of trees to visualize an immediate preliminary result, retraining the model with increasing numbers of trees to improve result quality over time.

3. DEMO PROPOSAL

To demonstrate the features of VIZDOM, we will use the Multiparameter Intelligent Monitoring in Intensive Care (MIMIC II) dataset [3], which contains clinical data for ICU patients. Figure 3 shows a simple task where a user trains a classifier to predict whether a particular patient has a metabolic disease (e.g., diabetes) given that patient's features (e.g., age, weight, blood pressure). In this example, the user can drag a logistic regression classifier from the list of available operators onto the screen (Step A). The user can then drag the classification label (metabolic) and attribute groups (all attributes) to the classifier and optionally tune the algorithm's hyperparameters (Step B). Once the user has supplied the necessary information, the computation starts in the background while the user immediately sees a classification summary, which includes the classifier's precision, recall, and F1-score (Step C). The user can then swipe the visualization to the left (Step D) to reveal a confusion matrix and ROC curve (Step E). All of these visualizations are updated incrementally as the computation continues, with the overall progress indicated by a bar at the bottom.

As shown by the summary statistics in Figure 3, the resulting classifier achieves only 66% average precision. In order to understand this relatively poor classification performance, the user can drill down to examine some of the statistics for individual attributes. The age histogram (Step F) shows that the dataset contains some pediatric patients that might be confusing the classifier. Therefore, the user can filter out these patients by using the pen to lasso only patients over the age of 20. Linking the age histogram with the logistic regression operator by drawing a line using the pen triggers a retraining of the classifier on only the selected subset of patients (Step G). Notice that the classification performance improves by excluding pediatric patients.

After identifying this subpopulation, the user might want to expand the binary classification task (i.e., metabolic diseases) to include multiple disease categories, as shown in Figure 4. This multi-label classification task builds an independent binary classifier for each of the specified labels. The example dataset includes 10 distinct disease categories (e.g., metabolic, infectious, heart failure), and the user can modify the existing logistic regression classifier by dragging

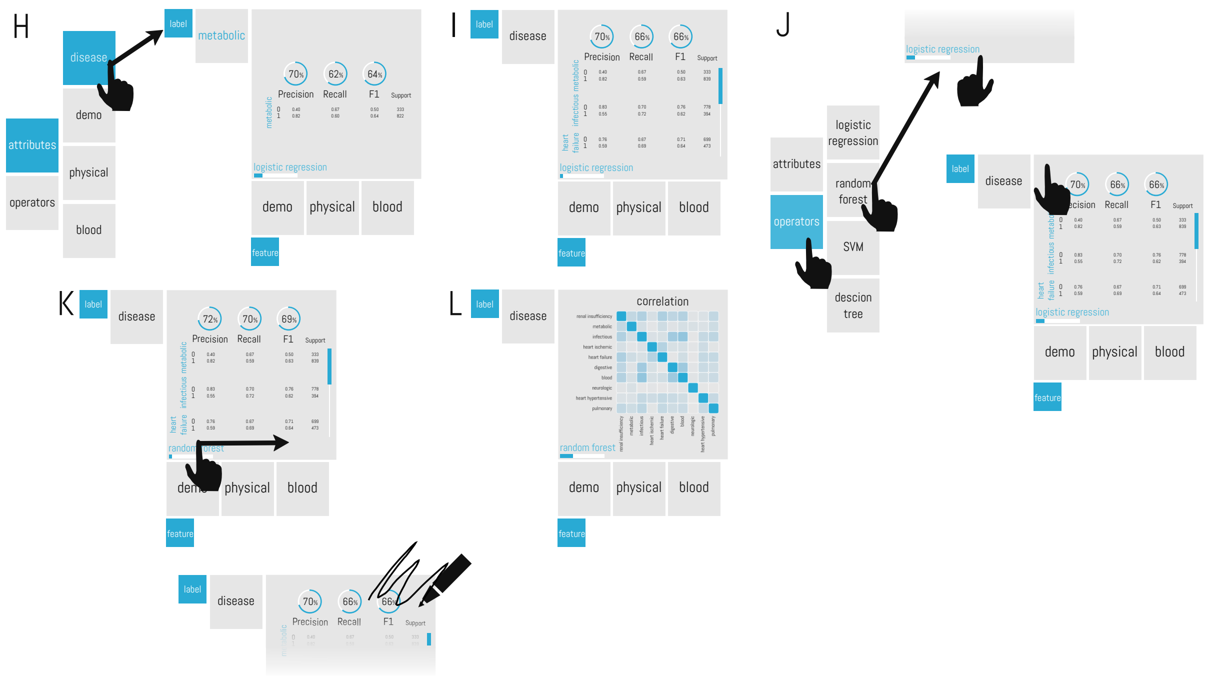

Figure 4: Story Board II

the entire disease group to the operator's label tab (Step H). The resulting visualization now includes a separate classification summary for each individual classifier (Step I).

While the logistic regression classifier is running, the user might want to concurrently train a different type of classifier in order to compare classification performance. A multi-touch gesture (i.e., holding the operator with one finger and tapping outside with another) creates a copy of the logistic regression operator, and the user changes the classifier to a random forest by dragging out the corresponding operator (Step J). As previously mentioned, all results are updated incrementally so the user can quickly compare classification performance on-the-fly and cancel suboptimal classifiers early by using a scribble gesture (Step K). The user can then swipe through the classification summaries for the remaining random forest classifier as shown previously in Steps D and E. Additionally, for all multi-label classification tasks, VIZDOM provides a two-dimensional histogram depicting the correlation between the outputs of individual classifiers (Step L). In this example, the visualization allows the user to identify comorbid diseases (i.e., diseases that frequently co-occur in patients).

The interactive exploration of the MIMIC II dataset represents the primary workflow of the demo, and we will also include free-form exploration over other real-world datasets.

4. ACKNOWLEDGMENTS

This research is funded in part by the Intel Science and Technology Center for Big Data, the NSF CAREER Award IIS-1453171, the Air Force YIP AWARD FA9550-15-1-0144, and gifts from SAP, Mellanox, and Oracle.

5. REFERENCES

[1] Apache Hadoop. `http://hadoop.apache.org`.

[2] Apache Mahout. `http://mahout.apache.org`.

[3] MIMIC II Dataset. `http://mimic.physionet.org`.

[4] RapidMiner. `http://rapidminer.com`.

[5] D. Antenucci, M. Cafarella, M. Levenstein, C. Ré, and M. D. Shapiro. Using Social Media to Measure Labor Market Flows. Technical report, 2014.

[6] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, U. Çetintemel, and S. B. Zdonik. Tupleware: "Big" Data, Big Analytics, Small Clusters. In *CIDR*, 2015.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, pages 10–18, 2009.

[8] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The MADlib Analytics Library or MAD Skills, the SQL. In *VLDB*, pages 1700–1711, 2012.

[9] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. MLbase: A Distributed Machine-learning System. In *CIDR*, 2013.

[10] Z. Liu and J. Heer. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.*, pages 2122–2131, 2014.

[11] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. E. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska. MLI: An API for Distributed Machine Learning. In *ICDM*, pages 1187–1192, 2013.

[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*, pages 15–28, 2012.

[13] E. Zgraggen, R. C. Zeleznik, and S. M. Drucker. PanoramicData: Data Analysis through Pen & Touch. *IEEE Trans. Vis. Comput. Graph.*, pages 2112–2121, 2014.

[14] C. Zhang, A. Kumar, and C. Ré. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*, pages 265–276, 2014.