Lowering the Barrier to Applying Machine Learning

Kayur Patel

[†]Computer Science & Engineering
DUB Group, University of Washington
Seattle, WA 98195
kayur@cs.washington.edu

ABSTRACT

Machine learning algorithms are key components in many cutting edge applications of computation. However, the full potential of machine learning has not been realized because using machine learning is hard, even for otherwise tech-savvy developers. This is because developing with machine learning is different than normal programming. My thesis is that developers applying machine learning need new general-purpose tools that provide structure for common processes and common pipelines while remaining flexible to account for variability in problems. In this paper, I describe my efforts to understanding the difficulties that developers face when applying machine learning. I then describe Gestalt, a general-purpose integrated development environment designed the application of machine learning. Finally, I describe work on developing a pattern language for building machine learning systems and creating new techniques that help developers understand the interaction between their data and learning algorithms.

Author Keywords

machine learning, software development

ACM Classification Keywords

H5.2 Information Interfaces and Presentation: User Interfaces; D2.6 Programming Environments: Integrated Environments.

General Term Human Factors

INTRODUCTION AND MOTIVATION

Machine learning is at the core of many advances in science and technology. Within HCI, researchers have applied machine learning to search [7], facilitating creativity [12] and helping people live healthier lives [4]. Within computer science, machine learning can reduce system downtime [2] and detect anomalous network behavior [3]. In humanity's greatest pursuits, machine learning can help to understand cancer [5] and the beginnings of the universe [1].

Despite the widespread impact of learning algorithms, ordinary software engineers seldom use these algorithms. One reason is that applying machine learning is difficult in ways *different* than traditional programming. Traditional programming is often discrete and deterministic, but most machine learning is stochastic. Traditional programming focuses on modules and lines of code, but machine learning focuses on pipelines and data. Traditional programming is often debugged with print statements and breakpoints, but machine learning requires analyses with visualizations and statistics. Traditional programming allows developers to *explicitly describe* the behavior of a program, but systems that use machine learning must *learn* behavior from data. Developers need new methods and tools to support the task of applying machine learning to their everyday problems.

The goal of my research is to support this programming with machine learning by understanding and alleviating the difficulties developers face when trying to use machine learning within software. In this paper, I review results from two studies that look at the difficulties developers face when using machine learning. I then proceed to describe integrated development environment built around addressing these difficulties. Finally, I describe ongoing work on creating new techniques that help developers debug machine learning systems and on distilling a pattern language for developing machine learning systems.

RELATED WORK

My thesis draws inspiration from recent work on tools that support the machine learning process for specific problem domains. For example, Crayons uses a coloring metaphor for training image segmentation classifiers [6]. Eyepatch allows composition and training of classifiers to create vision systems. Exemplar supports direction manipulation methods for specifying simple sensor-based recognize [9]. The domain-specific nature of such tools is both a strength and a weakness. Domain knowledge allows tools to limit the decisions required for a developer to create a system. But these same limitations then constrain the developer if a tool's assumptions do not match the developer's needs.

I also draw inspiration from general-purpose development environments, in particular environments like MATLAB. My work shows that people experienced in the application of machine learning report a preference for MATLAB, and it does provide better support than most programming environments. For example, in MATLAB, matrices are first-class objects and therefore a good fit for tabular data representations. Many learning algorithms involve solving linear algebra problems, also well supported by MATLAB. Finally, MATLAB makes it easy to analyze data by reducing

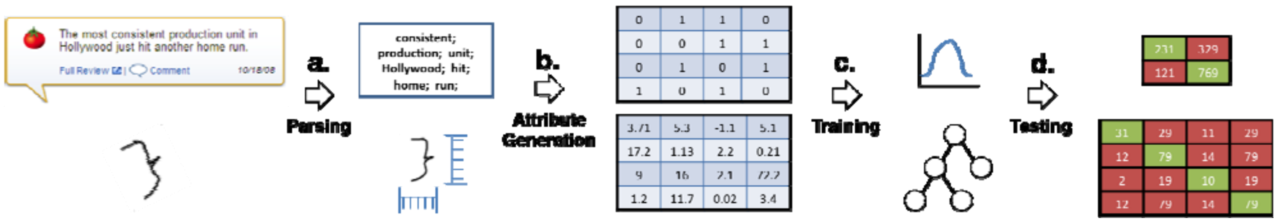
The most consistent production unit in Hollywood just hit another home run.
Full Review | Comment 10/18/08
a. Parsing
consistent; production; unit; Hollywood; hit; home; run;
b. Attribute Generation
c. Training
d. Testing

Figure 1: The figure above shows a common classification pipeline for two problems: movie-review sentiment analysis and pen-based gesture recognition. Developers must set up a working classification pipeline build a machine learning system. The pipeline consists of parsing data (a), generating attribute (b), training a model (c), and testing that model (d).

boilerplate code needed to sort, filter, and create basic visualizations.

Finally, my initial studies draw inspiration from prior work on understanding the difficulties that novice developers face learning how to program [8]. Experienced developers using machine learning are in a similar situation. Just as novice programmers have to learn a new way of thinking when learning programming, experienced programmers have to rethink how they program when dealing with systems that learn from data and provide stochastic results.

TWO STUDIES

Figure 1 illustrates a *classification pipeline* that is typical of many machine learning applications. Data must be collected in some raw format, which is processed to extract attributes, which are used to train a model, which is then evaluated in experiments. To inform our design of new tools and understand the current process developers take, I conducted two studies that examined the challenges developers encounter when using existing tools [11]. First, I interviewed eleven researchers who had built learning-based software. These researchers described and diagrammed their processes, discussing not only successful strategies but also pitfalls and difficulties. I interviewed researchers with machine learning expertise as well as researchers relatively new to machine learning who were applying it in their research. I reasoned this mixture of expertise would uncover difficulties that people encounter in applying machine learning as well as best practices for overcoming those difficulties.

My second study sought to further ground the results of my interviews through laboratory observations of actual work. Ten new participants each spent five hours building the machine learning component of a small application, a simple handwriting recognition engine. Participants provided input at all stages of the learning system: they collected data, wrote Java code to generate attributes, trained models using the Weka library [14], and conducted experiments to test the accuracy of their system.

From the results, I distilled three main difficulties that developers face when using machine learning. First, the studies show that the successful application of machine learning is generally based in an *iterative and exploratory* process. A developer examines all of the steps in the pipeline to find the step where they can make changes that will have the most impact on how well the entire system works. Second, the studies show that developers often have good intuitions about the individual links in their chain (e.g., their data and their features), but find it hard to *understand the relationship* between accuracy and these familiar parts of the machine learning system. Third, developers have difficulty *evaluating the learning system in the context of their application*. Developers are often concerned with more than just accuracy. For example, a developer of an embedded system may care about speed of attribute computation and classification (because this code needs to run on the embedded device), and might be willing to make tradeoffs in classification accuracy related to this performance.

GESTALT

Based on the initial studies, I built Gestalt: an integrated development environment that supports machine learning [10]. Gestalt addresses two of the difficulties described in the prior study. It helps developers iterate and explore by supporting a common machine learning process. And it helps developers understand relationships between data, attributes and classification results by providing connected, interactive visualizations of data. By addressing these difficulties, Gestalt fills a hole in current general-purpose tool support. Current general-purpose tools either help developers solve one step in the classification pipeline or they provide no support for machine learning. They do not support the machine learning process.

Supporting the Machine Learning Process in Gestalt

The machine learning process can be decomposed into two high level tasks: *implementing* of a classification pipeline and *analyzing* of data as it moves through the pipeline. Implementation a classification pipeline consists of gathering data and writing code to parse data, generate attributes, train a model, and test that model (Figure 1). Analysis consists of visualizing output from each step of the pipeline to understand the machine learning system. The process involves easily transitioning between implementation and analysis.

In Gestalt, developers interact with a classification pipeline in Gestalt through two high-level perspectives: an *implementation* perspective and an *analysis* perspective (Figure 2). This parallels the common distinction between coding and debug perspectives in modern development environments (e.g., Eclipse, Microsoft Visual Studio). Gestalt includes a number of capabilities that work together to help developers successfully use machine learning.

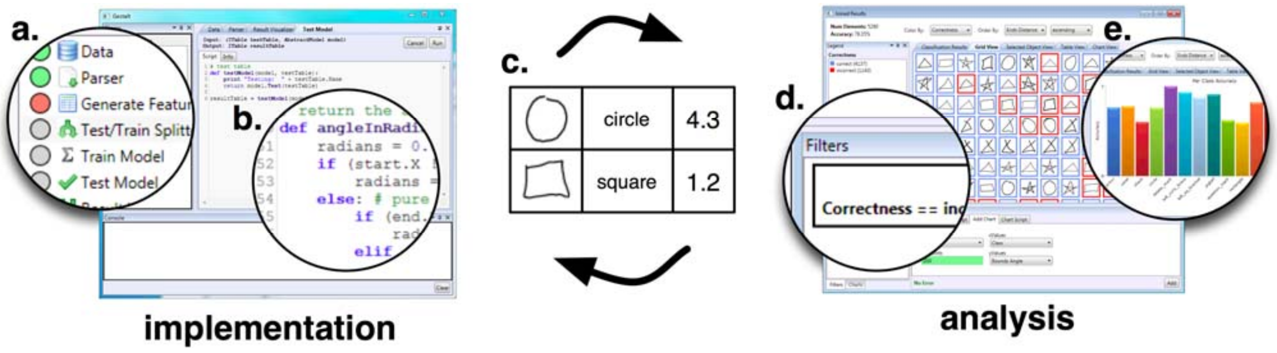

Figure 2: The implementation perspective provides developers with structure through its classification pipeline view (a) and flexibility by allowing them to write code to represent their specific problem (b). A common data structure (c), shared between analysis and implementation, allows developers to quickly switch between the two tasks. The analyzing allows developers to interact with the provided visualizations (e) by filtering, sorting, and coloring (d).

How do I represent my problem?

Representing a problem involves decomposing machine learning task into manageable steps. Gestalt helps developers effectively represent their problem by providing a structured representation of the classification pipeline. Gestalt provides general support through a *structured* set of explicit steps with standardized inputs and outputs (Figure 2a). Gestalt preserves *flexibility* by defining each step using IronPython scripts written in a built-in text editor (Figure 2b). This combination provides an explicit structure without constraining what a developer can do in that structure. Gestalt thus provides the same flexibility as general-purpose programming environments (e.g., Eclipse, MATLAB).

Where do I store my data?

The first step in the classification pipeline is loading data into an internal structure. Gestalt stores all information from the entire classification pipeline in a relational data table. Relational tables are a natural representation for discrete examples with many attributes. Because of this, they are also the backbone of many other general-purpose tools (e.g., Weka, Tableau). Gestalt differs from such tools because they do not address the entire classification pipeline (e.g., Weka focuses on a library of modeling algorithms, Tableau focuses on powerful visualizations of tabular data). Despite their common tabular nature, data representations in such tools are not identical. Developers using combinations of tools to address an entire pipeline must therefore explicitly attend to format conversion. The narrowed focus of each tool also means that information that could benefit analysis is often lost or unavailable when converting between tools. For example, Weka and other tools that represent examples as vectors of attributes generally lack support for examining the original data used to compute those attributes.

How do I see my data?

After loading data developers need to be able to inspect data to debug the behavior of the program. Developers reason about system behavior by examining data and its relationship to attributes and classification. Gestalt's support for many data types is enabled by a key distinction between *individual* and *aggregate* visualizations. It is impossible for a general tool to provide pre-packaged visualizations for all possible types of data. Gestalt instead supports data visualization by separating the logic needed to view *one example* from the logic to combine many single examples into an aggregate view. Developers can write code to visualize an example, and Gestalt then integrates that into aggregate visualizations throughout the pipeline.

How can I relate my data, attributes, and results?

Grouping and summarizing examples can help a developer understand a classification pipeline. Gestalt's analysis emphasizes *interactive visualizations*, inspired by work in interactive visualization tools [13]. Support is provided for faceted browsing, filtering, sorting, and coloring examples. Grouping and summarization operations can be applied according to attribute values, according to columns added to examples by steps in the classification pipeline, and according to tags added to examples by a developer.

Gestalt's support for machine learning goes beyond such prior general-purpose visualization tools by *connecting data generated across the entire classification pipeline*. In the case of domain-specific tools, consider that the coloring metaphor in Crayons is effective in part because it connects the pipeline's beginning (labeling data) and end (analyzing model classification) within a single visualization. Gestalt generalizes this with visualizations that connect data from different steps in the pipeline to help developers understand relationships between data, attributes, and results.

The "Gestalt" of Gestalt

Gestalts capabilities work together to better support machine learning. As a whole, these capabilities serve to accelerate the interactive loop: *developers can more quickly implement and analyze different potential versions of a machine learning system*. Gestalt's approach provides both structure and flexibility for rapid implementation, the shared data table removes data conversion and management to make it easy to switch between implementation and analysis, and connected visualizations allow developers to quickly analyze the important parts of their system.

Results

I ran a study evaluating how well developers found bugs in Gestalt. I compared Gestalt to a state-of-the-art baseline tool similar to MATLAB. Participants were asked to find bugs in existing solutions to two different machine learning problems (the problems in Figure 1). Participants unanimously preferred Gestalt and were able to find more bugs using Gestalt than using the baseline.

ONGOING WORK

My current work is focused on creating better techniques for understanding the relationships between data, attributes, and model results. Gestalt focuses on combining aggregate visualizations of individual examples and faceted browsing to help developers group examples. By comparing examples developers can leverage their intuition about the problem domain to understand the behavior of the system. However, the ways in which developers can group data within Gestalt are limited. One particular limitation is that Gestalt is subject to algorithmic biases.

Biases are harmful because they make it hard to understand if an example was misclassified because of a problem in the data or the attributes, or if it was misclassified because the peculiarities of one particular algorithm. To address algorithmic biases, I have been working on a system called Prospect. Prospect looks at the output of many different algorithms to help marginalize out the bias of one algorithm.

FUTURE WORK

I am still weighing options on next steps to finish my PhD. Gestalt addresses two of the difficulties involved faced by our developers: difficulty iterating and exploring and difficulty understanding relationships between data and results. I have not worked on support for helping developers evaluate the effectiveness of their machine learning systems. One direction for my research is to catalog a set of evaluation methods and come up with guidelines for when it is appropriate to use one method over another. Tools support can be built around facilitating a number of evaluation methods, instead of simply supporting random cross-validation.

In the same vein, creating a common pattern language for machine learning would be an invaluable tool for developers applying machine learning. Software design patterns provide developers with a set of common, scalable building blocks. Patterns allow developers to concentrate more new functionality rather than developing abstractions. Patterns are often extracted by looking at common structure across many different problems. Machine learning patterns go beyond the code, because building effective learning systems involves following best practices for collecting, processing and managing data. These practices are often offline and act more as guidelines for human behavior. Work focused on finding good software and human design patterns and understanding the interplay between these two patterns can ease the difficulties of applying machine learning.

DESIRED FEEDBACK

My dissertation work is approaching a point where feedback on how best to complete by PhD would be greatly beneficial. I will mostly likely be presenting my DC talk shortly before or after my official dissertation proposal. Consequently, there is room to change the future direction of my work based on feedback from the meeting.

I've also been working on framing my body of work. Most of my research focuses on making machine learning accessible to developers. While that is a strong common thread, I would appreciate critiques of my work, especially on the framing of the work and the larger vision.

REFERENCES

1. Ball, N.M. and Brunner, R.J. *Data Mining and Machine Learning in Astronomy*. 2009.
2. Candea, G. and Fox, A. Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel. *HotOS 2001*.
3. Chen, M.Y., Accardi, A., Kiciman, E., Lloyd, J., Patterson, D., Fox, A., and Brewer, E. Path-based Faliure and Evolution Management. *NSDI 2004*.
4. Consolvo, S., McDonald, D.W., Toscos, T., Chen, M.Y., Froehlich, J., Harrison, B., Klasnja, P., LaMarca, A., LeGrand, L., Libby, R., Smith, I., and Landay, J.A. Activity Sensing in the Wild: a Field Trial of Ubifit Garden. *CHI 2008*.
5. Cruz, J.A. and Wishart, D.S. Applications of Machine Learning in Cancer Prediction and Prognosis. *Cancer Informatics 2,* (2007).
6. Fails, J.A. and Olsen, D.R. Interactive Machine Learning. *CHI 2003*.
7. Fogarty, J., Tan, D., Kapoor, A., and Winder, S. CueFlik: Interactive Concept Learning in Image Search. *CHI 2008*.
8. Ko, A.J., Myers, B.A., and Aung, H.H. Six Learning Barriers in End-User Programming Systems. *VLHCC 2004*.
9. Maynes-Aminzade, D., Winograd, T., and Igarashi, T. Eyepatch: Prototyping Camera-based Interaction through Examples. *UIST 2007*.
10. Patel, K., Bancroft, N., Drucker, S.M., Fogarty, J., Ko, A.J., and Landay, J.A. Gestalt: Integrated Support for Implementation and Analysis in Machine Learning ProcessesNo Title. *UIST 2010*.
11. Patel, K., Fogarty, J., Landay, J.A., and Harrison, B. Investigating Statistical Machine Learning as a Tool for Software Development. *CHI 2008*.
12. Simon, I., Morris, D., and Basu, S. MySong: Automatic Accompaniment Generation for Vocal Melodies. *CHI 2008*.
13. Stolte, C. Visual interfaces to data. *SIGMOD 2010*.
14. Witten, I.H. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2005.