Main takeaways

- Data iteration in addition to model iteration;
- Diverging histogram to encode prediction correctness;
- Combine the tooltip for the two distributions of an overlaid histogram;
- Table for summary statistics + Histogram;
- Data blacklist/blocklist concept (or filtering logic bank);
- Sensitivity histogram/bar chart (the X-axis corresponds to the number of adjacent prediction changes across all data versions).

Understanding and Visualizing Data Iteration in Machine Learning

Fred Hohman[ϒ][*], Kanit Wongsuphasawat[α], Mary Beth Kery[χ], Kayur Patel[α]

[ϒ] Georgia Tech
Atlanta, GA, USA
fredhohman@gatech.edu

[α] Apple Inc.
Seattle, WA, USA
{kanitw, kayur}@apple.com

[χ] Carnegie Mellon University
Pittsburgh, PA, USA
mkery@cs.cmu.edu

ABSTRACT

Successful machine learning (ML) applications require iterations on both modeling and the underlying data. While prior visualization tools for ML primarily focus on modeling, our interviews with 23 ML practitioners reveal that they improve model performance frequently by iterating on their data (e.g., collecting new data, adding labels) rather than their models. We also identify common types of data iterations and associated analysis tasks and challenges. To help attribute data iterations to model performance, we design a collection of interactive visualizations and integrate them into a prototype, CHAMELEON, that lets users compare data features, training/testing splits, and performance across data versions. We present two case studies where developers apply CHAMELEON to their own evolving datasets on production ML projects. Our interface helps them verify data collection efforts, find failure estendem cases stretching across data versions, capture data processing changes that impacted performance, and identify opportunities for future data iterations.

Author Keywords

Data iteration, evolving datasets, machine learning iteration, visual analytics, interactive interfaces

CCS Concepts

•Human-centered computing → Visual analytics; •Computing methodologies → Machine learning;

INTRODUCTION

Successful machine learning (ML) applications require an iterative process to create models that deliver desired performance and user experience [4, 38]. As shown in Figure 1, this process typically involves both model iteration (e.g., searching for better hyperparameters or architectures) and data iteration (e.g., collecting new training data to improve performance). Yet, prior research primarily focuses on model iteration.

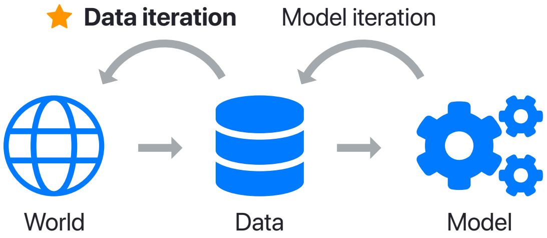


Figure 1. An overview of a typical machine learning process, which involves both model iteration (e.g., changing model architectures or hyperparameters) and data iteration (e.g., collecting new data to improve new model performance). This paper focuses on data iteration as its tooling is underexplored compared to model iteration.

Machine learning (ML) researchers are rapidly proposing new model architectures for tasks in computer vision and natural-language processing, in some cases producing weekly state-of-art results. This focus on modeling has lead to new emerging research areas. ML systems research supports more efficient and distributed training of multiples models simultaneously [2, 8, 10], and automated machine learning (AutoML) systems spawn hundreds of models at once and apply end-to-end model selection and training so that users need only input their data [12, 31]. Within the human-computer interaction and visualization communities, interactive machine learning and human-in-the-loop research often contributes new systems and interaction techniques to help model developers compare, evaluate, and understand models [4, 11, 40, 46].

This primary focus on model iteration makes sense in academic and research settings, where the objective is to build novel model architectures independent of which dataset is used. Yet in practice, the underlying dataset also determines what a model learns—regardless of which model or architecture is chosen. The classic ML colloquialism, *"garbage in, garbage out,"* evokes this essential fact that data needs to have the appropriate signal for a model be useful. In real world applications, rarely do teams start out with a dataset that is a high-quality match to their specific ML project goals. Thus *data iteration* is vital to the success of production ML projects.

To create high performance models, developers need to iterate on their data alongside their model architectures. Over the course of a machine learning project, datasets may change jointly alongside models for a variety of reasons. As a project matures, developers may discover use cases underrepresented in their datasets and thus need to collect additional data for such cases. Changes in the world may also affect the distribution of new data. For example, the latest viral video may drive spikes in internet search traffic and change search query distributions. These data changes raise interesting challenges and questions during model development: How does one track,

[*] Work done at Apple Inc.

CHI'20, April 25–30, 2020, Honolulu, HI, USA

ACM ISBN 978-1-4503-6708-0/20/04.

DOI: https://doi.org/10.1145/3313831.3376177

visualize, and explore a dataset changing over time? Is a certain model stable with respect to data change (e.g., does the performance improve or regress)? Does adding more data to an underperforming area of a model fix the problem?

In this paper, we focus on *data iteration* as a fundamental process in machine learning. To better understand data iteration practice and explore how interactive visualization can support data iteration, we make the following contributions:

- **Formative research on data iteration practice.** Through a set of interviews with 23 machine learning practitioners across 13 teams at Apple, we identify common types of data iterations as well as the tasks and challenges practitioners face with evolving data. Users: ML practitioners at Apple
- **Interactive visualizations for evolving machine learning datasets.** We design and develop a collection of interactive visualizations for evolving data that let users compare data features, training/testing splits, and performance across data versions. We integrate them into a prototype, Chameleon, to illustrate how the visualizations work together to help model developers explore data changes and attribute them to model performance.
- **Case studies on analysis of evolving datasets.** We present two case studies in which model developers apply Chameleon and its visualizations to examine their own evolving datasets used in machine learning projects. We find that our interface helps the model developers verify their prior data collection efforts, find failure cases that stretch across data versions, capture data processing changes that impacted model performance, and prompts them to perform future data iterations.

We hope this paper helps emphasize the importance of designing data as equally important as designing models in the ML process, inspiring future work around evolving data.

BACKGROUND & RELATED WORK

Our work draws upon and extends prior research within machine learning development and iteration literature, visual analytics for machine learning, and visual data exploration. Note that data *iteration* is subtlety different from data *processing*. Data processing describes mechanical transformations of static data (e.g., converting raw user logs to data tables) whereas data iteration is more concerned with the evolving process of how data changes during model development.

Evolving Machine Learning

Applying machine learning, particularly in production settings, often involves long and complex iterations [4, 29]. During these iterations, model developers must be careful that only one component of the modeling process changes to ensure fair comparison between trained models [27]. Enforcing this can be challenging, particularly when a particular model is integrated into a larger AI-system. This is the CACE principle, *"Changing Anything Changes Everything,"* in action: any one change during the modeling development process, from initial collection to monitoring after deployment, can have wide-reaching effects on final model performance [38]. These unique challenges also impact software development for machine learning [29]. A recent study of ML industry professionals found that *"discovering, managing, and versioning the data needed for machine learning applications is much more complex and difficult than other types of software engineering"* [3]. Data collection alone can be a bottleneck for production ML; for a survey see [35]. One system, Prospect, uses multiple ML models to help people diagnosis errors by understanding relationship between data, features, and algorithms [28]. Other work from the database community agrees that ML pipelines struggle with data understanding, validation, cleaning, and enrichment [30]. Data changes can exacerbate these challenges. Data drift occurs when data changes over time, causing predictions to become less accurate as features and labels slowly change in unforeseen ways. Many methods are developed to detect data drift [6, 7], while other work aims to compute a "data diff" to provide a concise, interpretable summary of the differences between two datasets [42].

In this paper, through a set of interviews with ML developers at Apple, we investigate ML iteration, focusing on data iteration, and identify common types of data iterations, as well as the tasks and challenges practitioners face with evolving data.

Visual Analytics for Machine Learning

Previous work demonstrates that interaction between people and machine learning systems enables collaboratively sharing intelligence [41]. Visual analytics has since succeeded in supporting machine learning model developers with a variety of modeling tasks [15, 23, 24, 36], such as model comparison [48] and diagnosis [19]. Yet many interactive systems focus on improving model performance. For example, Modeltracker is a visualization that eases the transition from model debugging to error analysis, a common disruptive cognitive switch during the model building process [5]. Squares extends these ideas and supports estimating common performance metrics in multi-class classifieres while displaying instance level distribution information necessary for performance analysis [34]. With respect to instance-level analysis, MLCube Explorer and Activis both enable interactive selection of instance subsets for model inspection and comparison [16, 17].

Visualization has also supported other data-centered tasks in ML, such as ensuring data and annotation quality [21, 22]. Closer to our work are FeatureInsight and INFUSE, two systems that focus on improving feature ideation and selection using visual summaries [9, 20]. FeatureInsight explores how visually summarizing sets of errors supports feature ideation by contributing a tool that helps ML practitioners interactively define dictionary features for text classification problems [9]. INFUSE helps analysts understand how predictive features are being ranked across feature selection algorithms, cross-validation folds, and classifiers, which helps people find which features of a dataset are most predictive in their models [20]. Both systems consider data iterations where given a particular dataset, explore how practitioners transform the dataset so that a model can capture the appropriate predictive signal. Both systems also help model developers transform their datasets, so that a model can capture the appropriate predictive signal. We contribute to visual analytics literature by designing and developing interactive visualizations that specifically support retrospective analysis of data versions and instance prediction sensitivity over time throughout ML development.

Visualization for Data Exploration

During exploratory data analysis [43, 14], users may be unfamiliar with their resources (e.g., data), uncertain of their goals, and unsure how to reach them. These processes involve browsing to gain an overview of the data or searching the data to answer specifics questions, all while refining their goals and potentially considering alternative approaches.

There are a number of prior visualization techniques for data exploration. Faceted browsing [47] lets users use metadata to filter subsets of data that share desired properties. The rank-by-feature framework [39] allows users to examine low dimensional projections of high-dimensional data based on their statistics. Facets [1] focuses on visualizing machine learning datasets, including the training and testing split, through two visualizations that help developers see the shape of each feature and explore individual observations. Profiler [18] uses data mining methods to automatically flag problematic data to help people assess data quality. Voyager [44, 45] blends manual and automated chart specification to help people engage in both open-ended exploration and targeted question answering.

However, these systems do not support temporal aspects of data (how data can change over time) and often only show univariate summaries. In this paper, we extend data exploration visualization techniques to machine learning datasets that change over model development time, including a new visualization for showing feature distributions that incorporates model performance and supports data version comparison.

UNDERSTANDING DATA ITERATION

We conducted a set of semi-structured interviews with 23 machine learning practitioners to understand their iterative development process. The participants, as listed in Table 1, include applied ML researchers, engineers, and managers across 13 teams at Apple. These conversations centered on machine learning iteration and lasted an hour on average. In this paper, we focus on data iteration, a common yet understudied aspect of ML development. From the interview data, we used a thematic analysis method to group common purposes, practices, and challenges of data iteration into categories [13]. We then iterated and refined these categories as we conducted more interviews. Throughout the paper, we use representative quotes from participants to illustrate the main findings from the study. We refer to the practitioners by their labels from Table 1.

Why Do Data Iteration?

Data Bootstraps Modeling

ML projects often start with a small amount of data and a simple model, and then scale to more data and more sophisticated model architectures during development (CV7, CV8). This approach allows developers to conduct lightweight experiments, making faster progress at the start of the project to test feasibility (S1). Upon starting a project, practitioners may not know specific characteristics of the data needed for the modeling tasks. Thus, they often start with publicly available datasets if possible, or gather a small amount of data to begin modeling. From there, if different types of data are required, an annotation task is designed and deployed to gather data and labels (CV4, CV5). In a scenario like this, newly annotated data can be highly valuable for informing modeling decisions and gauging the success of the project.

Domain	Specializations	Practitioners
Computer vision	Large-scale classification, object detection, video analysis, visual search	CV{1–8}
Natural language processing	Text classification, question answering, language understanding	NLP{1–8}
Applied ML + systems	Platform and infrastructure, crowd-sourcing, annotation, deployment	AML{1–5}
Sensors	Activity recognition	S1

Table 1. Interviewed ML practitioners, by domains and specializations.

assessing

Data Improves Performance

A more striking reason for conducting data iteration is what CV7 said when asked how to best improve model performance once a state-of-the-art model architecture has already been chosen and trained:

> "Most of the time, we improve performance more by adding additional data or cleaning data rather than changing the model [code]." —CV7, Applied ML developer

In this scenario, augmenting the existing dataset with new data instances is the preferred action to improve model performance and robustness compared to experimenting further on the model code and architecture. S1, a machine learning engineer working on a computer vision project, says that every month their team receives roughly 5% more labeled data that can have a significant effect on model performance. This type of data iteration, namely data collection (surveyed extensively in [35]), is frequently used in production ML projects, but occurs less often in traditional research settings.

The World Changes, So Does Data

Until now, we have discussed data iteration as an intentional process to improve model performance, but sometimes data change is imminent and out of the developers' control. Some practitioners said that their modeling procedure can be reactionary depending on changes in the world that impact the type, quality, and amount of collected data (NLP1, NLP5, NLP6, NLP7). This can have far reaching implications on model performance and user experience, particularly when new types of data that a model has never observed before are generated and collected.

Entangled Iterations

The practitioners often distinguished iterations on the model versus the data, but noted both are inherently intertwined over the course of a machine learning project's lifespan. Separating these two components to ensure fair comparisons (e.g., across models or data) is essential for making development progress. We also observed the subtle notion that data iteration, while fundamental to production machine learning development, can be buried in language while describing work and communicating modeling limitations. While describing a previous project, NLP4, a machine learning manager, caught himself mid-sentence to clarify that, *"over time means over the course of the data changing."* NLP8 explained that current tooling, including systems and visualizations, typically only gain investment for more established projects, where dashboards are developed to compare and track important metrics. These dashboards show models over time without explicitly separating model iterations from data iterations.

Common Ways Practitioners Iterate on Data

To understand how data can change, we first analyze the space of data evolution within machine learning. Consider the basic possible operations for how either the rows or columns of a dataset could change: + *Add*, − *Remove*, and ↻ *Modify*. These operations can be applied to each of the common components of a machine learning dataset (e.g., instances, features, labels) to enumerate possible data iterations.

Another common distinction within machine learning datasets compared to other types of data is the train/test/validation split. The three operations can also apply to each of these data subsets. This design space helps us enumerate possible data iterations broken down by common ML dataset components.

From our interviews, we recorded common data iterations taken in practice and which domain they most frequently occur. Below we list these, as well as their corresponding mapping in our evolving data design space.

+ *Add sampled instances*. **Gather more data randomly sampled from population.** In projects where data labels are already included in the raw data, e.g., where human annotation is not required, data iteration typically involves automatically collecting new instances at regular intervals from the user population and incorporating them into the modeling pipeline (S1, NLP6, NLP7, AML5).

+ *Add specific instances*. **Gather more data intentionally for specific label or feature range.** When certain instance types are known to be underrepresented, practitioners will intentionally target collection efforts to better balance the global data distribution. Practitioners mentioned this is useful when data labeling is particularly time consuming or requires significant effort. Ultimately this helps them get the best *"bang for the buck"* (CV3, CV4).

+ *Add synthetic instances*. **Gather more data by creating synthetic data or augmenting existing data.** CV7 and CV8 described two scenarios to improve the robustness of their models: synthetically creating new data (for 3D computer vision tasks), such as rendering 3D scenes in different lighting and camera positions, and augmenting datasets (such as rotating and translating images for computer vision tasks).

+ *Add labels*. **Add and enrich instance annotations.** Enriching existing data, for example, by adding more annotations to existing images, is a preferred approach when new raw data collection is unavailable or costly (AML4).

− *Remove instances*. **Remove noisy and erroneous outliers.** Removing and filtering undesired instances typically happens within processing or modeling code, taking the form of a *"data blacklist"* or *"filter bank"* (S1, NLP6, CV6).

↻ *Modify features, labels*. **Clean, edit, or fix data.** Data cleaning, a ubiquitous data iteration, is also incorporated into ML processing pipelines, and can be the focus of experiments to test its impact on performance (CV6).

We see that the most common operation used in practice is the addition of more data, features, and labels. Conversely, few data is ever removed. Modifying data encompasses a range of subtlety different processes, from cleaning messy data to editing existing labels to ensure high-quality annotations. These findings (1) corroborate existing work [35] that breaks down data collection into acquisition, labeling, and updating tasks, and (2) summarize how practitioners conduct data iteration.

Data Iteration Frequency

Regarding how often practitioners update their models or datasets, answers varied widely by project and domain. Some practitioners explained that their model changes monthly (NLP1, AML2), weekly (CV4), or as frequent as daily (AML2), where datasets change ranging from monthly (NLP5, S1, NLP6, AML5), weekly (NLP6, CV4, CV5), daily (NLP3), and even per minute (NLP1). While the rate at which a model updates correlates with how fast the data changes, the time that new data takes from being collected to being included in a new model may be on a different time scale. NLP4 expressed that the rate of which data iteration occurs depends on the project. He cited logistic constraints (e.g., the number of developers, budget, and annotation effort) as the causes of the variation.

Challenges with Data Iteration

When prompted about their general machine learning practice and if their datasets change over the duration of model development, nearly every practitioner voiced that they have experienced challenges (**C1–C5**) in their work.

Tracking Experimental and Iteration History (**C1**)

A particularly difficult challenge is keeping track of model performance across iterations. While helping people understand their model history has been explored in literature [27], understanding data history remains underexplored. *"Can't we go back to see how we used to be doing?"*, says NLP5, directly calling out a lack of model and data tracking tooling support. CV2 told us that many people simply look at the metrics, but that these can hide spurious subtleties. For example, even if the overall performance across versions remains constant, certain data subgroups may regress over time. Indeed, prediction churn, predictions for data instances that change given some change in the modeling pipeline, can occur given a new data iteration without any change to the model code (NLP5). To overcome this challenge, another engineer said a common method for comparing models trained on old data from weeks ago and a new model with new data is to fix the modeling code

Bang for the buck: efficiency/cost-effectiveness/value

https://en.wiktionary.org/wiki/bang_for_the_buck (e.g., architecture, hyperparameters), and retrain with the new training data, while fixing the testing sets (S1).

Similarly, practitioners want to know how their data changes across versions. NLP4 said they want to know how the inputs to their model change, since they consistently collect data to update their models. AML5 voiced similar concerns, and said it is important to understand data and feature drift during model development. AML5 also said they have implemented automatic methods to detect data drift [6, 7], which act as *"good sanity checks"* during development.

When to "Unfreeze" Data Versions (**C2**)

Choice of data period and its refresh

How does one make informed model decisions during persistent data collection? *"Fix [the data], and pray to god it doesn't change,"* said AML2, an experienced ML manager. AML3 corroborates this, noting that teams will freeze a window of data to tune model architectures. However, eventually this window must be expanded to account for new data. AML3 also said this window is often fixed longer towards a project's inception, but as real-world data is annotated or collected, the freeze time shrinks towards the end of development to be consistently evaluating against fresh data.

Regarding testing sets, CV5 and NLP8 emphasized their projects contain *"golden test sets:"* test sets that are usually hidden during development so practitioners prevent overfitting. These tests sets are usually fixed over longer periods of time (NLP7) to ensure wide coverage of evolving datasets; however, *these too* will eventually need to be updated to account for data drift and avoid overfitting to a specific golden set.

When to Stop Collecting Data (**C3**)

Modern ML models are data-hungry, but crowdsourcing annotations can take significant resources (e.g., time and money). Given the value that new data brings to a model, it is difficult to know when to stop data collection (AML4). CV4 said, *"we don't know when to stop getting data, so we start building [models] as soon as new data arrives."* This is corroborated in prior data collection work [35]. AML4, who works in crowdsourcing and data annotation, said as new data arrives, its value is high, but over time as a project's definition becomes more fixed and the data distributions solidify, collection may no longer be the top priority, depending on the project.

Manual Failure Case Analysis (**C4**)

intensificado

Since many ML projects start as experiments to prototype what is possible, robust software infrastructure is not an initial priority. In these scenarios, manual anomaly detection and error analysis can be time consuming yet critically important. This is heightened in projects where data can change, since incorrectly predicted instances may change from version to version (NLP5) [27]. NLP2 and CV5 said that during more fragile stages of a project, they *"work retrospectively, looking at specific fail cases to find patterns."* To find such error patterns, practitioners usually break down model metrics by class (or some other meaningful grouping) and list every mistake in hopes to identify a common thread.

Subgroup discovery/Data slicing/ Breakdowns

Building Data Blacklists (**C5**)

Practitioners explained that instances are usually removed from a dataset when they contain undesired and erroneous feature values or labels. These instances will either prevent a model from generalizing, or are deemed not relevant for a project's appropriate success. S1, NLP6, and CV6 noted that their projects contain a living list of instances to remove, i.e., a *"data blacklist"*. This bank of filtering logic continuously grows as data changes and is applied to raw data during processing stages to ensure the inputs to a model are high-quality.

VISUALIZING DATA ITERATION: MOTIVATION & TASKS

From the interviews, there is a clear desire for better tooling and interfaces for evolving data. Practitioners said that existing tools were either insufficient or nonexistent, and expressed enthusiasm for visualization tools to help them attribute data changes to model performance. These conversations also yielded several key ideas that inspired us to design new interactive visualizations for understanding data iteration.

As discussed in the interview findings, to analyze the effect of data changes on model performance, developers need to isolate the data changes from model architecture changes by fixing a model code constant while comparing data versions. We aim to design interactive visualizations to support this data comparison scenario. To inform our design, we distill tasks that practitioners need to perform to understand how data evolution affect model performance:

T1. Track and retroactively explore data iterations and model metrics over data versions (**C1**, **C4**).

T2. Attribute model metric change to data iterations (**C2**, **C3**).

T3. Compare feature distributions by training and testing splits, by performance (e.g., correct v. incorrect predictions), and by data versions (**C2**, **C3**, **C5**).

T4. Understand model sensitivity over data versions (**C4**, **C5**).

CHAMELEON: VISUAL ANALYTICS FOR EVOLVING DATA

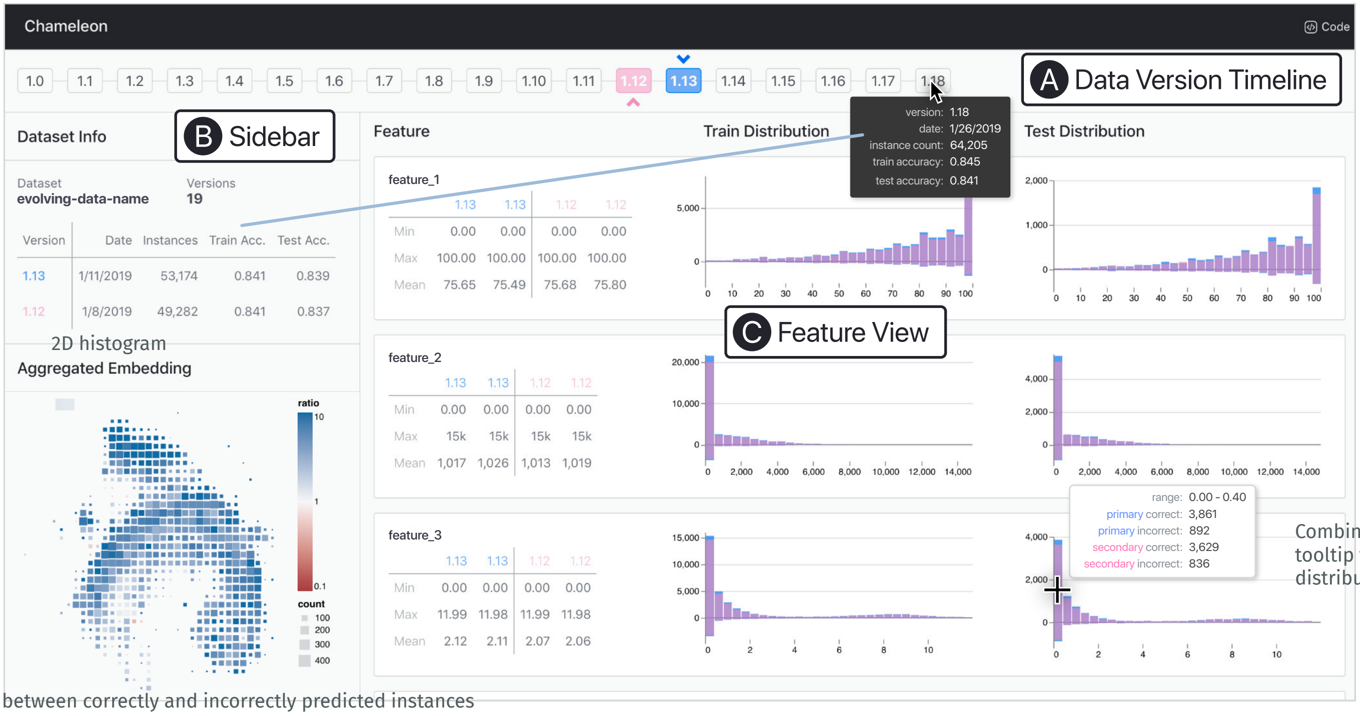
With the tasks identified from our formative research, we present a collection of interactive visualizations integrated into a prototype, CHAMELEON, that enables model developers to interactively explore data iteration during ML development (Figure 2): (A) the *Data Version Timeline*, (B) the *Sidebar*, and (C) the *Feature View*. We describe the implementation using tabular data; however, we have designed CHAMELEON such that extending to other data types only requires adding domain-specific views, e.g., an instance viewer to show images, text, or sensor streams. Throughout the following section, we link relevant views and features to the supported tasks (**T1**–**T4**).

Data Version Timeline: Data Iterations Over Time

To help practitioners track and inspect data versions, the top of the interface includes a *Data Version Timeline* (Figure 2A). To examine a particular version, users can click a blue arrowhead indicator (⌄) above a particular version in the timeline. To compare the primary selected data version with another version, users can select a secondary version by clicking the pink arrowhead indicator (⌃) below a particular version. To see details about a data version, hovering over any version displays a tootip including the version's date, the number of instances, and the model's train/test accuracy (**T1**).

Vega Tooltip

https://github.com/vega/vega-tooltip



Chameleon

Code

1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 1.10 1.11 1.12 1.13 1.14 1.15 1.16 1.17 1.18

A Data Version Timeline

version: 1.18
date: 1/26/2019
instance count: 64,205
train accuracy: 0.845
test accuracy: 0.841

B Sidebar

Dataset Info

Dataset: evolving-data-name

Versions: 19

Version	Date	Instances	Train Acc.	Test Acc.
1.13	1/11/2019	53,174	0.841	0.839
1.12	1/8/2019	49,282	0.841	0.837

2D histogram

Aggregated Embedding

Feature

Train Distribution

Test Distribution

C Feature View

feature_1

	1.13	1.13	1.12	1.12
Min	0.00	0.00	0.00	0.00
Max	100.00	100.00	100.00	100.00
Mean	75.65	75.49	75.68	75.80

feature_2

	1.13	1.13	1.12	1.12
Min	0.00	0.00	0.00	0.00
Max	15k	15k	15k	15k
Mean	1,017	1,026	1,013	1,019

feature_3

	1.13	1.13	1.12	1.12
Min	0.00	0.00	0.00	0.00
Max	11.99	11.98	11.99	11.98
Mean	2.12	2.11	2.07	2.06

range: 0.00 - 0.40
primary correct: 3,861
primary incorrect: 892
secondary correct: 3,629
secondary incorrect: 836

Combined tooltip for both distributions

Ratio between correctly and incorrectly predicted instances

Figure 2. The CHAMELEON interface integrates multiple coordinated views to help practitioners explore their evolving datasets. The *Data Version Timeline* (A) lists data versions across the top of the interface and allows users to select a primary and a secondary version to visualize below. The *Sidebar* (B) shows version summaries and multiples views that visualize changing instance predictions. Practitioners can use the sibebar views to filter data in the *Feature View* (C), which visualizes each feature of a dataset as a histogram with both selected data versions, faceted by performance and the train/testing split. Dataset and feature names are redacted for anonymity.

Redact: edit (text) for publication

Check the image on the next page

Feature View: Visualizing Evolving Distributions

The primary view of CHAMELEON is the Feature View (Figure 2C), which helps developers understand the distributions of all features in an evolving dataset. The view shows each feature as a row that contains summary statistics and *overlaid diverging histograms* for the training and testing splits. Given a pair of selected data versions, users can inspect how their datasets change with respect to the feature distributions, faceted by the training and testing split, as well as by the performance of other data versions.

Consider one feature's distribution for the training set (the test set distribution visualization is analogous) in Figure 3. Each distribution is represented as a histogram, where the x-axis encodes the feature's units and the y-axis is a count of the number of instances within a specific bin. However, we extend this chart by also including performance information: data instances that are correctly predicted are placed above the axis, where instances that are incorrectly predicted are placed below. This allows users to quickly see regions of the feature are underperforming (**T3**). However, given a new dataset based on some data iteration, is the new dataset's distribution the same as the old one? Extending this visualization design further, besides showing only the primary selected version in the Feature View, we overlay the secondary version on top, which allows users to now inspect a single feature's distribution and compare the distribution shape and performance across two data versions (**T3**). Hovering over any element of one of the overlaid diverging histograms displays a tooltip with details including the bin range, how many instances are included in the bin, and how many instances are correctly and incorrectly predicted for both the primary version and secondary version.

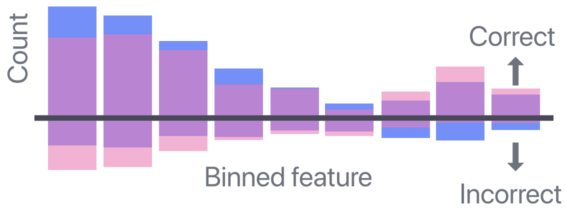
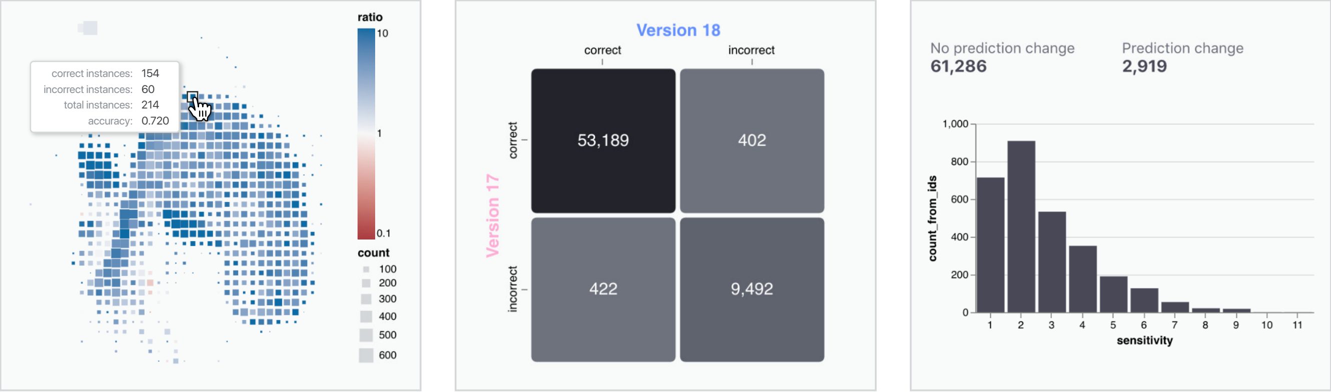


Figure 3. The *Feature View* visualizes each feature with an *overlaid diverging histogram*. The binned feature units are placed on the x-axis and the count on the y-axis. Data instances within a bin that are correctly predicted are included above the x-axis, and instances that are incorrectly predicted are included below the x-axis. We overlay the primary and the secondary dataset versions for version comparison.

The *overlaid diverging histogram* is a result from an iterative design process in which we aim to include the training/testing split, model performance, and data versions in a single view. As seen in prior work [5, 34], there are many ways to facet each feature. We overlay the histograms of both data versions so users can use the same axis to compare their distributions. We split the training and testing distributions into separate views since their scales often differ in magnitude (testing sets are usually much smaller than training sets). Lastly, we encode performance as a shift in y-axis so that users can compare parts of each distribution that are more often correctly (or incorrectly) predicted along a feature's unit axis.

It can be useful for detecting outliers too

Considering an ordered list of data versions



A. The ***Aggregated Embedding*** allows users to identify clusters of similar instances using dimensionality reduction.

B. The ***Prediction Change Matrix*** reveals how instance predictions change between a pair of data versions.

C. The ***Sensitivity Histogram*** shows prediction sensitivity (the number of prediction changes over all versions) across instances.

Figure 4. The Sidebar provides multiple views to visualize evolving instance predictions.

Sidebar: Visualizing Evolving Instance Predictions

The top of the Sidebar (Figure 2B) displays metadata about the selected data versions, including the metrics listed from the Data Version Timeline (**T1**). The Sidebar also contains three interactive visualizations that provide different views inside evolving ML datasets and their impact on the modeling process, including the Aggregated Embedding (Figure 4A), the Prediction Change Matrix (Figure 4B), and the Sensitivity Histogram (Figure 4c). Clicking any visual element within any of the three visualizations filters the Feature View. Below we discuss the usage and encodings of each visualization.

Aggregated Embedding

The Aggregated Embedding (Figure 4A) shows the primary data version via a discretized summary plot of the data's dimensionality reduction output. This plot gives users an overview of the dataset and is useful for discovering potential clusters of similar instances (**T2**).

For reduction and visualization, the Aggregated Embedding uses UMAP [26]: a non-linear dimensionality reduction that better preserves global data structure compared to other techniques like t-SNE [25]. UMAP often provides a better "big picture" view of high-dimensional data while simultaneously preserving local neighbor relations. Since datasets can contain many instances in practice, we use a 2D histogram (instead of a scatterplot) to visualize the embeddings. Essentially, we discretize the embedding space and aggregate instances into cells. Each cell size is scaled by the number of instances within that cell. To indicate the performance of the instances in each cell, we also color cells using a diverging color-scale based on the ratio between correctly and incorrectly predicted instances.

To identify clusters, users can look for a number of different patterns, such as large blue cells (many instances that are mostly correctly predicted), smaller red cells (fewer instances that are mostly incorrectly predicted), and groups of similar cells (**T2**). Hovering over any cell displays its details, including the number of correct instances, incorrect instances, their sum (the size encoding), and its prediction performance ratio (the color encoding). Clicking on any cell filters the Feature View to show its instances in feature space.

Prediction Change Matrix

Confusion matrix

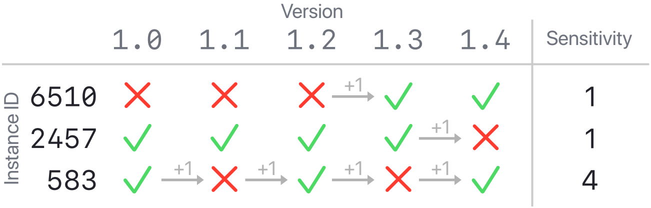
When training on new data, does a model improve in the expected ways, or does it unexpectedly regress? Given a pair of data versions (primary and secondary), the Prediction Change Matrix (Figure 4B) shows the subset of the instances that exist in both versions and facets them by both their prediction correctness and version (**T4**). As a result, the four cells within the Prediction Change Matrix include:

- *Previous version correct, current version correct.* Instances unaffected by the data iteration (Figure 4B, top-left).
- *Previous version incorrect, current version incorrect.* Instances require more attention to improve the model (Figure 4B, bottom-right).
- *Previous version incorrect, current version correct.* Instances where the data iteration improved the model and caused existing data to become correctly predicted (Figure 4B, bottom-left).
- *Previous version correct, current version incorrect.* Instances where the data iteration caused the model to regress (Figure 4B, top-right).

Cells are colored and labeled with their corresponding instance count. To help model developers understand where these groups of instances are located within the dataset, clicking any cell filters the Feature View to show where in each feature's distribution the instances are located (**T2**).

Sensitivity Histogram

The Sensitivity Histogram (Figure 4C) shows the prediction sensitivity of data instances over versions within the selected version range (**T4**). Given a range of data versions in an evolving dataset, we compute the following sensitivity measure: for each data instance, obtain its prediction using the new model trained on each data version; then, given this ordered list of predictions, compute the number of adjacent changes the predictions make. For example, at an early version of the dataset,



Instance ID	1.0	1.1	1.2	1.3	1.4	Sensitivity
6510	✗	✗	✗ +1 →	✓	✓	1
2457	✓	✓	✓	✓ +1 →	✗	1
583	✓ +1 →	✗ +1 →	✓ +1 →	✗ +1 →	✓	4

Figure 5. An instance's sensitivity denotes the number of its prediction changes throughout data versions. Instance 6510 has a sensitivity of 1, changing from incorrectly to correctly predicted (model improvement). Instance 2457 also has a sensitivity of 1, but flips from correctly to incorrectly predicted (model regression). Lastly, instance 583 has a sensitivity of 4, indicating that the instance is sensitive to model changes.

a particular data instance may be incorrectly predicted by a model, but a later model trained with more data collected may correctly make a prediction for the instance. We compute this measure for each instance within an evolving dataset, and plot its distribution in the Sensitivity Histogram. Figure 5 shows this computation on three example data instances: one that improves model performance, one that causes a regression, and another whose prediction constantly flips.

The Sensitivity Histogram lets users see how data instance predictions change over data versions. The x-axis shows the sensitivity measure (number of prediction changes across data versions) while the y-axis shows its count. Users can find data instances whose predictions may change frequently across versions, hinting that these types of instances may be close to the model's decision boundary or underrepresented within the entire dataset (**T4**). Conversely, many data instances may never change their prediction across data versions, indicating that these instances are not impacted by the taken data iteration. This view could help developers identify types of instances that need to be considered separately from the global model or suggest further data collection of instances with different feature values. Lastly, clicking on any bar within the Sensitivity Histogram filters the Feature View to show the distribution of these instances across the dataset's features (**T2**).

System Design and Implementation

CHAMELEON is a modern web-based interface built with React[1] and Redux[2]. All data processing[3] and model training[4] was done in Python using the standard data science suite. Visualizations were constructed and rendered using Vega-Lite [37].

CASES STUDIES ON DEPLOYED ML PROJECTS

We demonstrate CHAMELEON on two case studies where invited model developers used the system to analyze their own evolving machine learning datasets used in products at Apple. For each use case, we loaded the developer's evolving dataset into CHAMELEON. The developer then spent 1½-hours analyzing their data. During the sessions, they verbalized their thought process in a think-aloud protocol.

To start, each practitioner signed a consent form and provided background information about their project and ML iteration practice. After a system tutorial, we let them explore their evolving dataset using CHAMELEON, while encouraging them to verbalize their insights, actions, concerns, and any developed hypotheses. We prepared a set of general questions based on CHAMELEON's capabilities to guide practitioners in case they were unsure what to view next. Examples include: Do the feature distribution shapes change over time in expected ways? Can you find any instances that change predictions often? Where do they lie in feature space? Are there any clusters of underrepresented points you did not know about?

Each session ended with an exit interview that asked practitioners to reflect on their process of jointly iterating on their data and model during project maturity, their experience using our interactive interface, and if it could be useful for them.

We aim to simulate ML development where a model is affected by a changing dataset. Our setup is as follows: given the modeling code that creates a model from a given dataset, we fix this code (e.g., architecture, hyperparameters). Now, given an evolving dataset that has changed over time, we apply the modeling code to each dataset version, training a new model at every version, so that a series of models exists that can be compared fairly with respect to data versions. We also account for non-determinism in the model's training process by fixing a random seed. This setup is the inverse of the model iteration process, where a dataset is frozen and one iterates on the model. Note, while the evolving datasets used in both case studies are real, the models here were trained by the researchers and do not reflect production metrics, e.g., accuracy. To preserve anonymity, dataset and features names are redacted.

Case Study I: Sensor Prediction

The first project aims to classify sensor data from mobile phones to decide whether to engage a particular feature on device. The dataset has 64,502 instances from real users, collected over 2 months. Each instance has 20 features, most of which are quantitative values that represent a particular measurement quantized at various time windows (e.g., 1, 12, and 24 hours). The model developer, named D1 for anonymity, was actively working on improving their ML pipeline.

Visualization Challenges Prior Data Collection Beliefs

D1 first checked the overall performance of the model by inspecting the train and test accuracies, as well as the color of the Aggregated Embedding, noting that color the embedding was largely shades of blue, indicating a reasonably performing model. D1 then surveyed the entire list of features, discussing the overall shape of each feature. In this dataset, most features were skewed towards the end of the features' range. D1 explained that he was aware of this skew, but CHAMELEON made it much clearer, and ultimately he was surprised about magnitude of the skew (e.g., Figure 2C and Figure 6).

In fact, D1 said he was *"continually distracted"* by this skew throughout his session. He was constantly hypothesizing the behavior of the phone users' collected data. For example, D1 said, *"I am surprised [test users] behave like this, more than I originally thought."* Using the visualizations reinforced future analysis he wanted to run on a subset of the data within a particular feature range.

[1] React.js: https://reactjs.org

[2] Redux.js: https://redux.js.org

[3] pandas: https://pandas.pydata.org

[4] scikit-learn: https://scikit-learn.org/stable/

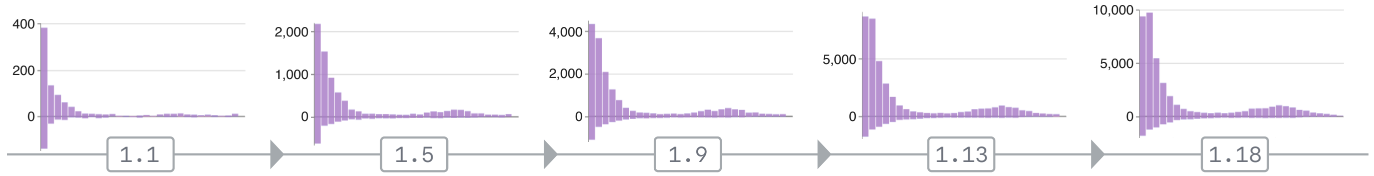


Figure 6. A feature's long-tailed, multi-modal distribution shape solidifies over collection time, from 1,442 (ver. 1.1) to 64,205 (ver. 1.18) instances.

Finding Failure Cases

From there, D1 used the Aggregated Embedding to filter the Feature View, searching for clusters of data that were either not performing well (e.g., red in the Aggregated Embedding) or had small number of instances (e.g., small in the Aggregated Embedding). D1 suggested that these are likely underrepresented subsets of users. Another group of instances D1 found did not have much *"history"* in the data (e.g., lacking richness in historical features), making it hard for the model to predict.

When inspecting all the previous data versions, D1 noted that the *"shape of the features solidifies quickly; in this dataset, that's a good thing to confirm"* (Figure 6). D1 explained that their team knows there exist outlier cases, but that they can only gain value from them by understanding their place in the context of the global distributions. D1 note that using CHAMELEON's filters mimics their manual error analysis, which includes selecting a specific set of filters and scrolling through instance by instance to *"get a feel for the error cases."* With these cases identified, the team can either exclude them on device or improve the model performance by considering to treat a particular subset of the data differently. D1 also noted that their models are purposefully tuned to be conservative, that is, favor one outcome over another, so that failure cases do not cause confusion for their phone users.

Interface Utility

D1 found the Aggregated Embedding the most useful and *"surprisingly fun,"* and thought the Sensitivity Histogram was a novel metric to find outliers he had not considered before. He said these two views filtering the Feature View is helpful for identifying instances to remove from the dataset (informing and conducting a − `Remove instances` iteration). D1 also said CHAMELEON is most useful in two scenarios: (1) general data exploration tasks, especially when one does not define data collection nor featurization, and (2) situations where his team *"expands our user base, where data can change quickly."*

Case Study II: Learning from Logs

The second project uses interaction logs from mobile phones to decide whether to recommended particular actions to a phone's user. The dataset has 48,000 instances from real users, collected over six months. Each instance has 34 features, most of which are quantitative values that are normalized to represent a phone user's application behavior. The model developer, named D2 for anonymity, is also actively working on improving their machine learning pipeline.

Inspecting Performance Across Features

D2 started their analysis by observing the global trends of the features in the Feature View. An immediate observation is that a handful of the features were power-law distributed. D2 said the long tails on these features are typically interesting and difficult instances to predict. After using the Aggregated Embedding to filter down to these long tails, D2 said, *"having the ability to click on [a cluster] and see a subset of your dataset across the features with their performance, that's really awesome"* (e.g., Figure 7). One of clusters identified was determined to have predictive power in a single feature, an insight that D2 said he did not know previously. *"I want to understand what the predictive features are, and this is a great way to do that."* After finding some more outlier groups that were harder to correctly predict (as seen by the diverging histograms), D2 noted that the model on these groups was still *"somewhat of a coin flip,"* and needed further analysis.

a.k.a. "long tail"

Capturing Data Processing Changes

As D2 explored older data versions, he noticed the model performance started high, and over versions, slowly decreased as more instances were included into the dataset. This is in stark contrast to the observed behavior in the previous case study, where D1's data collection steadily increased the model's performance. D2 found one of the larger performance drops. He then checked the date of the version and was excited to see that CHAMELEON had revealed a specific point in time that D2's data processing code had changed on device. As production ML projects depend on coordination between many different teams sourcing model inputs, knowing when an external change could have impact on a model is important. Thus, point events (e.g., software updates) and continuous feature drift should be captured and handled.

Encouraging Instance-level Analysis

Using CHAMELEON, D2 describe the mindset the interface placed him in when conducting his analysis. *"I'm drawn to start thinking about [instance-level analysis], which I don't normally do. Normally, it takes a lot of work to get these visualizations."* This is a important perspective shift, where a deep understanding of ones data, as opposed to aggregate statistics,

or instances

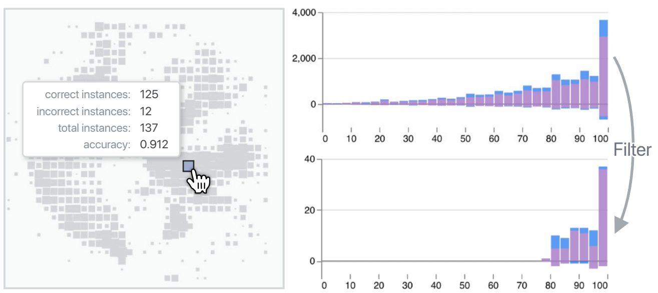


Figure 7. Filtering across features helps practitioners quickly find interesting data subsets to compare against the global distributions, e.g., this subset found using the Aggregated Embedding that performs well (91.2% accuracy) and has feature values strictly greater than 75.

is promoted in the machine learning process. D2 also discussed that the examples along decision boundary, found from the most sensitive instances using the Sensitivity Histogram, need to be understood more, and suggested cleaning the data or reconsidering how labels are applied, two possible future data iterations (↻ `Modify features`, ↻ `Modify labels`).

Case Study Discussion

We designed and developed CHAMELEON to investigate visual analytics tools for data iteration in production ML development. From our case studies, we learned how our interface, and more generally better data iteration tooling, could help model developers attribute data changes to performance. In particular, the model developers were enthusiastic about the interactivity of the visualizations, such as the different filters included that were often used during their analysis.

Improving Data Iteration Interfaces

Our developers surfaced specific features that deployed data iteration tooling should support, such as comparing datasets not only over time but by other metadata, such as their originating source. Others requested features included more options to manipulate single features, such as displaying their true labels for context, toggling y-axis normalization across train/test splits, and computing and ranking correlations between filtered feature distributions. The only confusion encountered using the visualizations was an initial distribution encoding interpretation mistake where D2 thought the diverging x-axis was the label and not performance.

Informing and Prompting Future Data Iterations

Lastly, it was interesting to see the model developers suggest new data iterations based on their analysis of the visualizations. Both D1 and D2 cited targeted collection constraints due to privacy concerns, yet they suggested other iterations (as listed earlier) that they thought would improve model performance. This tight loop of using interactive visualization to (1) show data iterations, and then (2) inform future data iterations demonstrates the power that data wields in machine learning.

OPPORTUNITIES FOR FUTURE DATA ITERATION TOOLS

Interfaces for Both Data and Model Iteration

While we focused on understanding data iteration within applied machine learning, there exists great opportunity for interfaces that combine both model and data iteration visualizations. In our interface, there is nothing unique about the Data Version Timeline that could not also include model iterations as an additional type of version. Moreover, there likely exists model iteration specific visualizations that could be included in the Sidebar that would display when a model iteration is selected. As many ML projects jointly iterate and develop their data and modeling pipeline, supporting any iteration, whether it be on the data or model, presents a future where any ML iteration can be tracked, visualized, and attributed to model metrics.

Data Iteration Tooling to Help Experimental Handoff

Transferência

In many large-scale ML projects, it is likely new people will join (or leave) over a project's lifespan. Since most ML projects go through extensive iteration where experimental records are not easily searchable nor understandable, future work may ease experimental handoff by providing an interactive history of the data iterations, models, and conducted experiments. Automatically tracking and externalizing this metadata surrounding ML practice could help newcomers become familiar with a project, since this information is typically only known by the people conducting the work.

Data as a Shared Connection Across User Expertise

Our work reinforces that successful ML projects require a deep inspection and exploration of the data used to create a model. Beyond expert ML practitioners that we focus on in this paper, there are many more people without ML expertise who wish to apply ML in their work. Regardless of ones' ML expertise and modeling approach, machine learning needs data. For novices, data iteration is also the only type of ML iteration that can be done without a significant investment to learn sophisticated ML algorithms. Thus, this data-first view of ML development is an important concept that helps both novice and expert users build better models.

"Label iteration" Think of Snorkel

Visualizing Probabilistic Labels from Data Programming

As data programming, weakly-supervised probabilistic label generation for large sets of unlabeled data [32, 33], becomes more popular and is used as an alternative to expensive data labeling tasks, it is important to know when and how label updates are made given a new data labeling rule. In this case, the data iteration is the changes of the labels for possibly many instances at once. Attributing data labeling rules to how labels change could help developers generate high-quality labels.

Limitations: Visualizations for Other Data Types

Our formative research and design goals apply broadly to evolving ML datasets. Our practitioners from diverse ML domains generally expressed a lack of support for data iteration in their current tools, suggesting that practical tooling for data iteration is an important and open area of research. CHAMELEON focuses on iterations of tabular data; however, ML is applied within many other domains where data takes different forms, such as images in computer vision or text in natural language processing. Future work should investigate visualization designs for data iteration in these data domains.

CONCLUSION

In this paper, we explored the practice of data iteration in production machine learning. We conducted formative research through interviewing 23 applied ML developers across 13 teams at Apple. We identified a set of tasks and challenges practitioners face to tackle changing datasets. We then designed and developed a set of interactive visualizations integrated into a prototype tool that supports these tasks. We demonstrated the tool's effectiveness on two case studies where model developers applied the interface to their own evolving datasets used in production ML projects. We hope this work emphasizes the importance of designing data as equally important as designing models in the ML process, inspiring future research and tooling around evolving data.

ACKNOWLEDGMENTS

We thank Donghao Ren and our colleagues at Apple for their time, effort, and help integrating our research with their work.

REFERENCES

[1] 2017. Facets. *Google PAIR* (2017). `https://pair-code.github.io/facets/`

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and others. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[3] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE Press, 291–300.

[4] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Magazine* 35, 4 (2014), 105–120.

[5] 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 SIGCHI Conference on Human Factors in Computing Systems*. ACM, 337–346.

[6] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, and Bernhard Pfahringer. 2017. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software* 127 (2017), 278–294.

[7] Aurélien Bellet, Amaury Habrard, and Marc Sebban. 2013. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709* (2013).

[8] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 65.

[9] Michael Brooks, Saleema Amershi, Bongshin Lee, Steven M Drucker, Ashish Kapoor, and Patrice Simard. 2015. FeatureInsight: Visual support for error-driven feature ideation in text classification. In *IEEE Conference on Visual Analytics Science and Technology*. IEEE, 105–112.

[10] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, and others. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*. 1223–1231.

[11] Jerry Alan Fails and Dan R Olsen Jr. 2003. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*. ACM, 39–45.

[12] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*. 2962–2970.

[13] Graham R Gibbs. 2007. Thematic coding and categorizing. *Analyzing Aualitative Data* 703 (2007), 38–56.

[14] Jeffrey Heer and Ben Shneiderman. 2012. Interactive dynamics for visual analysis. *Queue* 10, 2 (2012), 30.

[15] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. 2018. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Transactions on Visualization and Computer Graphics* (2018).

[16] Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. 2018. Activis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 88–97.

[17] Minsuk Kahng, Dezhi Fang, and Duen Horng Polo Chau. 2016. Visual exploration of machine learning results using data cube analysis. In *Workshop on Human-In-the-Loop Data Analytics*. ACM.

[18] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*. ACM, 547–554.

[19] Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. 2010. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1343–1352.

[20] Josua Krause, Adam Perer, and Enrico Bertini. 2014. INFUSE: Interactive feature selection for predictive modeling of high dimensional data. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1614–1623.

[21] Shixia Liu, Gennady Andrienko, Yingcai Wu, Nan Cao, Liu Jiang, Conglei Shi, Yu-Shuen Wang, and Seokhee Hong. 2018a. Steering data quality with visual analytics: The complexity challenge. *Visual Informatics* (2018).

[22] Shixia Liu, Changjian Chen, Yafeng Lu, Fangxin Ouyang, and Bin Wang. 2018b. An interactive method to improve crowdsourced annotations. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 235–245.

[23] Junhua Lu, Wei Chen, Yuxin Ma, Junming Ke, Zongzhuang Li, Fan Zhang, and Ross Maciejewski. 2017a. Recent progress and trends in predictive visual analytics. *Frontiers of Computer Science* 11, 2 (2017), 192–207.

[24] Yafeng Lu, Rolando Garcia, Brett Hansen, Michael Gleicher, and Ross Maciejewski. 2017b. The state-of-the-art in predictive visual analytics. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 539–562.

[25] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

[26] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[27] Kayur Patel. 2010. Lowering the barrier to applying machine learning. In *Adjunct Proceedings of the 23nd Annual ACM symposium on User Interface Software and Technology*. ACM, 355–358.

[28] Kayur Patel, Steven M Drucker, James Fogarty, Ashish Kapoor, and Desney S Tan. 2011. Using multiple models to understand data. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

[29] Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 667–676.

[30] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. Data management challenges in production machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1723–1726.

[31] Yao Quanming, Wang Mengshuo, Jair Escalante Hugo, Guyon Isabelle, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).

[32] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* 11, 3 (2017), 269–282.

[33] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *Advances in Neural Information Processing Systems*. 3567–3575.

[34] Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D Williams. 2017. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 61–70.

[35] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2019. A survey on data collection for machine learning: A big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[36] Dominik Sacha, Michael Sedlmair, Leishi Zhang, John A Lee, Jaakko Peltonen, Daniel Weiskopf, Stephen C North, and Daniel A Keim. 2017. What you see is what you can change: Human-centered machine learning by interactive visualization. *Neurocomputing* 268 (2017), 164–175.

[37] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2016), 341–350.

[38] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine learning: The high interest credit card of technical debt. *Google* (2014).

[39] Jinwook Seo and Ben Shneiderman. 2005. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization* 4, 2 (2005), 96–113.

[40] Patrice Y Simard, Saleema Amershi, David M Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, and others. 2017. Machine teaching: A new paradigm for building machine learning systems. *arXiv preprint arXiv:1707.06742* (2017).

[41] Simone Stumpf, Vidya Rajaram, Lida Li, Weng-Keen Wong, Margaret Burnett, Thomas Dietterich, Erin Sullivan, and Jonathan Herlocker. 2009. Interacting meaningfully with machine learning systems: Three experiments. *International Journal of Human-Computer Studies* 67, 8 (2009), 639–662.

[42] Charles Sutton, Timothy Hobson, James Geddes, and Rich Caruana. 2018. Data diff: Interpretable, executable summaries of changes in distributions for data wrangling. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2279–2288.

[43] J.W. Tukey. 1977. *Exploratory data analysis*. Addison-Wesley Publishing Company. `https://books.google.com/books?id=UT9dAAAAIAAJ`

[44] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2015. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2015), 649–658.

[45] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017a. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2648–2659.

[46] Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. 2017b. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2017), 1–12.

[47] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. 2003. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 401–408.

[48] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. 2019. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning Models. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 364–373.