Ease. ML: A Lifecycle Management System for Machine Learning

Author(s):
Aguilar Melgar, Leonel; Dao, David; Gan, Shaoduo; Gürel, Nezihe M.; Hollenstein, Nora; Jiang, Jiawei; Karlaš, Bojan; Lemmin, Thomas; Li, Tian; Li, Yang; Rao, Susie; Rausch, Johannes; Renggli, Cedric; Rimanic, Luka; Weber, Maurice; Zhang, Shuai; Zhao, Zhikuan; Schawinski, Kevin; Wu, Wentao; Zhang, Ce

Publication date:
2021

Permanent link:
https://doi.org/10.3929/ethz-b-000458916

Funding acknowledgement:
184628 - EASEML: Toward a More Accessible and Usable Machine Learning Platform for Non-expert Users (SNF)
187132 - Machine#based Scoring of a Neuropsychological Test: The Rey#Osterrieth Complex Figure (SNF)
957407 - Integrated Data Analysis Pipelines for Large-Scale Data Management, HPC, and Machine Learning (EC)

Ease.ML: A Lifecycle Management System for MLDev and MLOps

Leonel Aguilar[†], David Dao[†], Shaoduo Gan[†], Nezihe Merve Gurel[†], Nora Hollenstein[†]
Jiawei Jiang[†], Bojan Karlas[†], Thomas Lemmin[†], Tian Li[*], Yang Li[†,+], Susie Rao[†]
Johannes Rausch[†], Cedric Renggli[†], Luka Rimanic[†], Maurice Weber[†], Shuai Zhang[†]
Zhikuan Zhao[†], Kevin Schawinski[§], Wentao Wu[‡], Ce Zhang[†]
[†]ETH Zurich, [‡]Microsoft Research, Redmond,
[*]Carnegie Mellon University, [§]Modulos AG, [+]Peking University
firstname.lastname@{[†]inf.ethz.ch, [‡]microsoft.com, [§]modulos.ai, [*]cmu.edu, [+]pku.edu.cn}

ABSTRACT

We present Ease.ML, a lifecycle management system for machine learning (ML). Unlike many existing works, which focus on improving individual steps during the lifecycle of ML application development, Ease.ML focuses on managing and automating the entire lifecycle itself. We present user scenarios that have motivated the development of Ease.ML, the eight-step Ease.ML process that covers the lifecycle of ML application development; the foundation of Ease.ML in terms of a probabilistic database model and its connection to information theory; and our lessons learned, which hopefully can inspire future research.

1. INTRODUCTION

The wide application of machine learning (ML) technologies has spawned intensive and extensive research on ML systems. The increasing complexity of ML systems, however, has perplexed many application developers, especially those who are domain experts, but do not have much of a background in statistics and/or computer sciences. Although many of these users are proficient DBMS users because of its fascinating usability and clean abstraction, they are struggling with modern ML systems. We believe that an emerging challenge of wider applications of ML techniques hinges on the *usability* of ML systems for such non-expert users.

Recently, there has been a flurry of work focused on ML usability. Researchers have developed systems that make it increasingly easy to handle different components of the ML development process. Examples include data acquisition with *weak supervision* (e.g., Snorkel [19], ZeroER [34]), *debugging and validation* (e.g., TFX [16, 3], "Query 2.0" [35], Krypton [17]), *model management* (e.g., Cerebro [18]) and *deployment* (e.g., ModelDB [33], MLFlow [37]), *knowledge integration* (e.g., DeepDive [38]), *data cleaning* (e.g., HoloClean [21], ActiveClean [14]), and *interaction* (e.g., NorthStar [13]). Thanks to these efforts, today's users are well-equipped with many powerful tools for different stages of ML application development. However, there are still challenges in ML usability. One hypothesis (and observation) we have is that

> *One emerging barrier hindering the usability of ML systems is not the limited availability, capacity, and performance of existing ML tools, but rather the complexity for a non-expert user to navigate a collection of ever-updating, overwhelmingly powerful tools.*

Formulating this hypothesis is not trivial for us. It is the result of a four-year long effort of collaboration with academic users such as astronomers, biologists, social scientists [28, 31, 26, 30, 1, 29, 4, 5, 2, 27], and with industrial partners via our spinoff company, Modulos.ai. By observing the challenges and struggles that these non-expert users have when using existing ML systems, our view on ML usability has gradually shifted from emphasizing the *speed and efficiency* of ML training and inference, to the *capacity of automation* of ML training, *before* finally developing into our current view of the *lifecycle/process management* and the aforementioned hypothesis.

As a reflection of our hypothesis, we present Ease.ML, a research prototype aimed at lifting the burden of managing the entire development lifecycle from *non-expert* ML application developers. Unlike existing AutoML work that aims to provide automated tools for *individual steps* in the development cycle and other work that aims at making each individual component perform more easily and efficiently, the goal of Ease.ML is to provide automated *toolchains* and well-defined *processes* that *stitch together* existing/upcoming tools to improve end-to-end development experience. Ease.ML defines an *eight-step, human-in-the-loop process* to provide systematic guidelines for the users during the entire *ML journey*. Each of these steps often requires us to formulate new research problems and non-trivial technical solutions. Putting them together requires us to rethink factors such as the data model and execution model.

In this paper, we present the Ease.ML process and design decisions that we made in the model for data, execution, and user interaction. This forms the main contribution of this paper. Although the techniques to enable some version of each of the eight steps have been described before in different venues [25, 20, 24, 11, 8, 6, 7, 36, 15, 12, 22, 10, 23, 9], it is the first time that these components have been put together to form an integrated system. By presenting such a platform, we hope to stimulate discussion and inspire future research by exposing the *limitations* and *mistakes* that we made when developing Ease.ML.

2. EASE.ML PROCESS

The goal of the Ease.ML process is to provide systematic guidance for non-expert users building ML applications — by following each step in the Ease.ML process, users end up with an ML appli-

11th Annual Conference on Innovative Data Systems Research (CIDR '21) January 10-13, 2021, Chaminade, USA.

cation with certain quality guarantees. The goal of the system is to minimize human efforts in a way that is as systematic as possible.

The current version of the Ease.ML process contains eight steps, organized into three subprocesses. This process starts before an ML model is even constructed (we call it "Day 0" — *Pre-ML* Subprocess), and it continues through the DevOps phase after an ML model is constructed (we call it "Day 2" — *Post-ML Subprocess*). In this section, we take the perspective of an end user, and walk through a full user experience following an Ease.ML process.

Target User Profile. In Ease.ML, we assume the following profile for a non-expert user:

1. The user understands the domain, application, and dataset well. She knows how to clean up the data, where to acquire new data, and how to measure the success and quality of an ML model. The user also knows the specific task that she wants to achieve by using ML.
2. The user knows how to write a simple Python script to manipulate and transform her data, if necessary.
3. The user knows how to invoke an AutoML system by following the instructions or user manual.
4. The user knows basic concepts of ML (e.g., the meaning of accuracy and a classifier); however, she does not have experience in constructing real-world ML applications beyond the simplified examples she can find in a textbook.

2.1 Pitfalls and Confusions

Each step of Ease.ML is designed to avoid some of the pitfalls and confusions that users often struggle with in our experience, even when they are equipped with state-of-the-art ML tools. Before we present the concrete Ease.ML process, we first describe the key pitfalls and confusions that we hope to address. Note that, this list is *by no means* a complete list of struggles that users have when using existing ML ecosystems and *by no means* does Ease.ML provide a full solution to the listed issues.

P1. Unrealistic expectations of quality.

A common problem is that many users often have unrealistic expectations of the quality that ML can provide — it is not uncommon that our users come to us with a very noisy or ill-defined dataset, but still hope for >90% accuracy. Without identifying these problems as early as possible, many users are set on a journey of AutoML and feature engineering that is doomed to fail.

As an analogy to traditional software engineering, users need the functionality of automatic *feasibility study* for ML workloads.

P2. "How many training/testing examples do I need?"

A common question that many users ask during our first meeting is about the amount of data that they need to acquire. Systematic answers to such questions are important to ensure that users are working with representative data samples. Among these, a key step to ensure the quality of the development and deployment process is to *automatically* make sure users have a test set that is large enough to be representative, but small enough to be affordable to acquire. Another question, which most users do not explicitly ask, but is somehow more important, focuses on how to *manage* the statistical power of a test set. Even if given a large enough test set at the beginning, it can quickly lose its statistical power during testing and can lead to overfitting. This can often cause overly optimistic expectations on accuracy and misguide the development process.

As an analogy to software engineering, users need the functionality of *principled test-driven development support* for ML workloads.

P3. "What should I do to further improve accuracy?"

It is rare for an AutoML system to directly output a model with satisfactory quality — often, it is an iterative process in which users need to continuously improve the data artifact via feature engineering, data acquisition, data labeling, and data cleaning, and to run AutoML systems over different versions of the artifact. One confusion that users often have is *what to do next to further improve a data artifact?* Often, we see users who have access to a collection of powerful tools such as (1) Snorkel for weak label acquisition, (2) HoloClean for automatic data cleaning, (3) Label Box for label acquisition, (4) manual feature engineering, and (5) manual data cleaning, but nonetheless ask: *which tool should I use next?* It is not uncommon for us to see users spend a huge amount of time on one of these steps, but not end up with better ML accuracy.

As an analogy to software engineering, users need support to avoid *"pre-mature optimizations"* for ML workloads.

P4. "Will this new model that I just found on arXiv today work better?"

ML is an area that is undergoing rapid refreshing — everyday, there seem to be new methods developed for a specific application. One common question that users often have is whether these new techniques, developed by different researchers, can end up helping their ML tasks. However, maintaining an ML artefact with respect to new models and new methods can be expensive (in terms of both computation and manual efforts), and this is an area in which many users are looking for a principled solution.

As an analogy to software engineering, users need the support of *"continuous integration and delivery"* for ML workloads.

P5. "How can we know that we are not applying a model trained on February's data during Christmas?"

We see a subset of users (especially those from industry who are dealing with mission critical applications) who are continuously worried about the mismatch between incoming production data and models. Often, these users have access to a collection of models, each of which is constructed under certain assumptions on the production workload — for example, one may employ twelve models within a one-year time frame, each dealing with a single month's workload. The challenges that these users are facing are (1) how to automatically adapt to the ever-changing production data distribution; and (2) how to pick the best model to use given fresh production data in a labor-efficient manner.

The design of the Ease.ML process is inspired by these pitfalls and confusions, which we observed from interacting with our users.

2.2 Day 0: Pre-ML Subprocess

At the beginning of the process, the user provides Ease.ML with a dataset $\mathcal{D}$. We define the data model precisely in the next section but, intuitively, a dataset $\mathcal{D}$ consists of: (1) Four sub-datasets with similar schema: a labeled training set and validation set; and an unlabeled pool example and an unlabeled test set; (2) All the uncertainties — for example, missing feature value and all their candidate values, different possible weak supervision labels, and the results of different automatic data cleaning tools — are marked down as meta-data. As we will see, the above will be modeled using a data model that is a probabilistic database, which we will describe later.

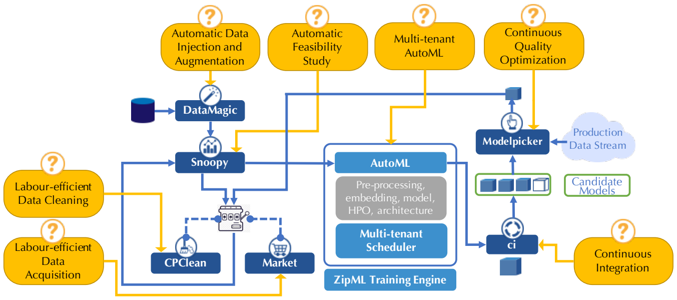
Automatic Data Injection and Augmentation
Automatic Feasibility Study
Multi-tenant AutoML
Continuous Quality Optimization
DataMagic
Snoopy
AutoML
Pre-processing, embedding, model, HPO, architecture
Multi-tenant Scheduler
ZipML Training Engine
Modelpicker
Production Data Stream
Candidate Models
ci
Labour-efficient Data Cleaning
Labour-efficient Data Acquisition
CPClean
Market
Continuous Integration

Figure 1: Ease.ML Process

The goal of the Pre-ML subprocess is to make sure that the *user systematically improves the data artefact until the system expects that ML can provide the level of quality that the user desires*.

2.2.1 *Automatic Data Injection and Augmentation*

- **Inspiring Pitfall/Confusion:** N/A
- **Input:** Input dataset.
- **Output:** Augmented, machine readable, dataset.
- **Next Step:** Automatic Feasibility Study.
- **Ease.ML Components:** Ease.ML/DataMagic [25, 20]

As the first step of the process, the user needs to get her data into the system. Data injection is often a painful process that is frequently overlooked, especially for data stored in a diverse format. As an important preliminary step, Ease.ML contains a set of default templates to deal with data injection. One such example is the `Document` template, for which Ease.ML translates documents stored in different formats, such as `.PDF`, `.DOCX`, and `.HTML`, into a single, machine-readable data structure. It also contains a set of default data augmentation templates that the systems can automatically apply in order to increase the data volume. In this step, the systems also automatically fire up existing automatic data cleaning and weak supervision tools, each of which provides one candidate value for uncertain features or labels.

2.2.2 *Automatic Feasibility Study*

- **Inspiring Pitfall/Confusion:** P1
- **Input:** (1) Augmented, machine readable, dataset; (2) Expected accuracy.
- **Output:** A {`Yes`, `No`} signal of whether the system believes that the expected accuracy is achievable given the input dataset.
- **Next Step:** If `No`, Labor-efficient Data Cleaning and/or Labor-efficient Data Acquisition; If `Yes`, AutoML.
- **Ease.ML Components:** Ease.ML/Snoopy [24]

Once the data is machine readable, Ease.ML conducts an automatic feasibility study — just like how a real-world ML consultant works, Ease.ML automatically looks at the data and provides its *belief* of the best possible accuracy that an ML model can provide. If the answer is "No," Ease.ML advises the user to improve her data artifact before firing up an expensive AutoML process.

Challenges.. To enable such an *automatic feasibility study* functionality, the system needs to estimate the *irreducible error* of a given task, known as Bayes error, in ML. Providing a practical Bayes error estimator has been an open problem in ML for decades.

2.2.3 *Labor-efficient Data Cleaning*

- **Inspiring Pitfall/Confusion:** P3
- **Input:** Augmented, machine readable, dataset.
- **Output:** A prioritized list of dirty data examples to be manually cleaned.
- **Next Step:** Automatic Feasibility Study.
- **Ease.ML Components:** Ease.ML/CPClean [11]

If the answer returned by the automatic feasibility study component is "No," Ease.ML advises the user to improve her data artifact before firing up an AutoML engine. To this end, the user has two options: (1) clean up the data manually or (2) acquire more data and labels. Ease.ML treats each of these two options as an "arm" in a multi-armed bandit setting and automatically balances the efforts spent on each of them. When the system asks the user to clean-up the data manually, it provides the user with a prioritized list of dirty examples. This list is selected by estimating the *impact* of cleaning up a certain data example to the downstream ML prediction accuracy. The goal is to minimize the manual cleaning effort to reach high ML accuracy. The system then reruns the *Automatic Feasibility Study* step and gets a new estimate on the irreducible error.

Challenges.. To enable such a *labor-efficient data cleaning* functionality, the system needs to analyze the impact of data cleaning over downstream ML tasks. Towards this goal, ActiveClean [14] has done seminal work for a subset of ML models. However, how to support this for more general ML models remains an open question.

2.2.4 *Labor-efficient Data Acquisition*

- **Inspiring Pitfall/Confusion:** P3
- **Input:** Augmented, machine readable, dataset.
- **Output:** A prioritized list of data examples to be manually labeled.
- **Next Step:** Automatic Feasibility Study.
- **Ease.ML Components:** Ease.ML/Market [8, 6, 7]

When the system asks the user to acquire more data, it provides her with a list of unlabeled data examples to look at. This list is selected by estimating the *importance* of acquiring labels for these unlabeled examples to impact the accuracy of downstream ML prediction accuracy. Given this list, the user can label these examples manually. The goal is to minimize the manual labeling effort to reach high ML accuracy. The system then reruns the *Automatic Feasibility Study* step and get a new estimate on irreducible error.

Challenges.. To enable such an *labor-efficient data acquisition* functionality, the system needs to provide a notion of "value" to a given data point with respect to the ML model. This is an emerging topic, related to data market, ML explainability, and active learning.

2.3 Day 1: AutoML Subprocess

2.3.1 *Multi-tenant AutoML*

- **Inspiring Pitfall/Confusion:** P4
- **Input:** Augmented, machine readable, dataset.
- **Output:** An "endless" *stream* of ML models.
- **Next Step:** Continuous Integration.
- **Ease.ML Components:** Ease.ML/AutoML [36, 15, 12]

Once the user passes the Pre-ML subprocess, she now has a data artifact that the system *believes* to have the potential to reach her expected accuracy. At this stage, Ease.ML fires up the AutoML component and generate ML models. Like other AutoML systems, Ease.ML deals with the entire end-to-end ML pipeline, including automatic feature selection and engineering, model selection, hyperparameter tuning, etc. The functionality of this step largely follows existing platforms, but with one specific twist — when there is a new ML model made available to the system, *all* users' applications that could potentially use this model are rerun. As a result, the outcome of the AutoML component of Ease.ML is not a single model, but a *stream* of models.

Challenges.. To enable such a *continuously updated* AutoML component, the system needs to be efficient in how it prioritizes different applications and different users. When there are multiple applications, each of which can be updated with latest model, the system needs to pick those applications that can benefit the most. This poses a new challenge that we call *multi-tenant AutoML*.

2.4 Day 2: Post-ML Subprocess

2.4.1 Continuous Integration

- **Inspiring Pitfall/Confusion:** P2
- **Input:** (1) A stream of ML models output by the AutoML component; (2) user-defined requirements of deployment.
- **Output:** A stream of ML models that satisfy the requirements.
- **Next Step:** Continuous Quality Optimization.
- **Ease.ML Components:** Ease.ML/ci [22, 10, 23]

The Post-ML subprocess concerns the deployment of models. Given a stream of ML models output by the AutoML component, not all models can be deployed. The first step in the Post-ML subprocess is for Ease.ML to test whether each of the models satisfies a user-defined deployment criteria, such as *the new model must be better than the old model by at least 5%* or *the new model cannot change more than 10% of the old model's predictions*. The system then only keeps those models that satisfy the given conditions. In Ease.ML, all these criteria can be tested by using a test set. The system lets the user know the number of test examples it needs to achieve a rigorous statistical significance level and when it needs a new test set to avoid overfitting.

Challenges.. The challenge is how to re-use a test set efficiently to provide rigorous guarantees, without overfitting. Intuitively, at the moment that Ease.ML provides the test result to a user, it "leaks" some information from the test set to the user. To avoid overfitting, the system needs to "measure" the information leakage and then automatically manage the statistical power of the test set. This is a challenging statistics problem known as adaptive analytics.

2.4.2 Continuous Quality Optimization

- **Inspiring Pitfall/Confusion:** P5
- **Input:** (1) A set of ML models that satisfy the user-defined requirements; (2) A set of unlabeled, fresh test examples from the production.
- **Output:** A set of test examples to label such that one can pick the best ML model to use in production.
- **Next Step:** Automatic Quality Debugging and Recommendations.
- **Ease.ML Components:** Ease.ML/ModelPicker [9]

In principle, all models passing the previous step can be candidates to be deployed in production. This gives us a candidate set of models. The goal of this step is to pick, from this candidate set, the best model to use for the next period of time (e.g., a day). To achieve this goal, Ease.ML asks the user to collect a set of unlabeled test examples from production (e.g., at 8 a.m. everyday). It then tries to pick the candidate model that performs the best on this fresh test set, by acquiring as few labels as possible. This chosen model is then used (e.g., the next day). The user repeats this process, e.g., on a daily basis, such that one often uses the best model for the ever-changing test data distribution. Ease.ML also automatically applies existing domain adaptation techniques, which create more models automatically — the system feeds them into the *continuous integration* step to make sure that they satisfy the deployment requirement and automatically add them to the candidate set.

Challenges.. The challenge is to minimize the number of labels that we need to acquire to distinguish between a large collection of models. This itself is an interesting online learning problem.

2.4.3 Automatic Quality Debugging and Recommendations

- **Inspiring Pitfall/Confusion:** P3
- **Input:** (1) Model deployed in production; (2) Mistakes that this model makes in production.
- **Output:** Recommendations of next steps to further improve this model.
- **Next Step:** Labor-efficient Data Cleaning and/or Data Acquisition.
- **Ease.ML Components:** N/A

At the end of the process, Ease.ML collects errors made by the currently deployed model (picked by the previous step), and gives users recommendations about how to further improve it. This step has the same structure as that after *automatic feasibility study* and directs the user to *labor-efficient data cleaning*, *labor-efficient data cleaning*, and/or *labor-efficient data acquisition*.

3. EASE.ML SYSTEM DESIGN

In this section, we describe the high-level design of Ease.ML. We focus on the design of the logical layer and omit the design of the physical layer due to space limitation.

Unified, Principled, yet Practical Framework for Human Interaction in ML. One key realization that we have, only after we constructed each of the individual components, is that all user-facing components can be modeled via the same data model — a *probabilistic database* in which uncertainty is induced by dirty data, unlabeled examples, weak supervision, etc. Moreover, all human interactions can be modeled as the process of *mutual information maximization*, i.e., eliminating uncertainties to maximize the "closeness" to an unknown, "ground-truth" world. This unified framework is often expensive to implement, and one often needs to develop non-trivial proxies and approximations for each individual component. Note that this particular view definitely has its own limitations, which we will discuss in the next section. Nevertheless, it does provide a principled, yet practical, framework to model human interactions in an end-to-end ML process.

3.1 A Probabilistic Data Model

The logical data model of Ease.ML is *a probabilistic database [32] in which each row has independent uncertainty*. This is our key design decision, which provides a unified abstraction for (1) noises in features and labels; (2) weak supervision (e.g., Snorkel) that provides multiple alternatives to labels; (3) availability of multiple state-of-the-art data cleaning tools that provide multiple candidates of the cleaned value; (4) unknown label values for data and label acquisition; and (5) data augmentation. This design decision was entirely unclear to us at the beginning — only after we put all components together, we realized how probabilistic databases would provide a clean, unified abstraction for almost all Ease.ML steps.

Logical Data Model. Let $\mathcal{X}$ be the feature domain and let $\mathcal{Y}$ be the label domain. Note that, despite its name, the label domain does not need to be categorical as in supervised classification. For example, both domains can be images for an image translation task. A *dataset* contains pairs of *random variables*, taking values in the feature domain and the label domain, respectively:

$$\mathbf{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : \mathbf{x}_i \in \mathcal{X}, \mathbf{y}_i \in \mathcal{Y}\},$$

and it induces two probability distributions

$$\Pr_{\mathbf{D}}[\mathbf{x}_i = X], \quad X \in \mathcal{X}; \qquad \Pr_{\mathbf{D}}[\mathbf{y}_i = Y], \quad Y \in \mathcal{Y}.$$

As a concrete example, $\mathbf{x}_i$ can be the output of multiple different state-of-the-art automatic data cleaning tools, each of which forms one candidate value that $\mathbf{x}_i$ can take for the i^{th} data example and $\Pr_{\mathbf{D}}[\mathbf{x}_i = X]$ is our prior on their relative accuracy. $\mathbf{y}_i$ can be the output of multiple labeling functions, each of which forms one candidate label that $\mathbf{y}_i$ can take for the i^{th} data example and $\Pr_{\mathbf{D}}[\mathbf{y}_i = Y]$ is our prior confidence on each of the candidate labels. Note that, in our setting, an *unlabeled dataset* is simply one in which $\Pr_{\mathbf{D}}[\mathbf{y}_i = Y]$ is a uniform distribution. One particular direction we take in Ease.ML is that there exists a single, *unknown ground truth configuration* for each uncertainty example $(\mathbf{x}_i, \mathbf{y}_i)$, and the *ground truth dataset* $G^*_{\mathbf{D}}$ is

$$G^*_{\mathbf{D}} = \{(x^*_i, y^*_i) : (\mathbf{x}_i, \mathbf{y}_i) \in \mathbf{D}\}$$

where (x^*_i, y^*_i) is the ground-truth value of $(\mathbf{x}_i, \mathbf{y}_i)$. In practice, the ground-truth dataset corresponds to the clean dataset if the uncertainty is induced by dirty data, or true labels if the uncertainty is induced by unlabeled examples and weak supervision.

Input to Ease.ML. The input to Ease.ML is four datasets, all with the *same* feature domain and label domain: (1) **Labeled Training Set $\mathbf{D}_{tr}$**, dataset to train an ML model – both features and labels have uncertainty; (2) **Pool of Unlabeled Data Examples $\mathbf{D}_{unl}$**, pool of data examples that are unlabeled and can be acquired to be included into the training set – features have uncertainty and labels have uniform prior (unlabeled); (3) **Labeled Validation Set $\mathbf{D}_{val}$**, validation dataset that can be used during development – features and labels *do not* have uncertainty, i.e., we have access to $G^*_{\mathbf{D}_{val}}$; (4) **Unlabeled Production Test Set $\mathbf{D}_{prod}$**, test set that comes from production – only labels have uncertainty (unlabeled).

3.2 ML Model

We apply ML training only over a deterministic dataset, which is the assumption of most ML models. A deterministic dataset is one without uncertainty: $D \subseteq \mathcal{X} \times \mathcal{Y}$. Given a deterministic dataset D, the training process of ML returns an ML model that is a mapping $\mathcal{A}_D$: $\mathcal{X} \mapsto \mathcal{Y}$. When we apply the training process over an uncertain dataset $\mathbf{D}$, it returns a random variable, $\mathcal{A}_{\mathbf{D}}$, taking values over all possible classifiers. The inference process of a given model $\mathcal{A}_D$ is to apply such a mapping to another deterministic dataset D': $\mathcal{A}_D(D')$. We use similar notation for datasets $\mathbf{D}$ and $\mathbf{D}'$ with uncertainties, in which $\mathcal{A}_{\mathbf{D}}(\mathbf{D}')$ returns a random variable, depending on the randomness in $\mathbf{D}$ and $\mathbf{D}'$.

3.3 Semantics of Each Step

In Ease.ML, we implement the end-to-end process described in the previous section by developing seven individual components. In this section, we describe the semantics of each step.

3.3.1 Uncertainty Introduction and Elimination

The first collection of Ease.ML components introduce and eliminate uncertainties involved in the dataset, including DataMagic (2.2.1), Snoopy (2.2.2), CPClean (2.2.3), Market (2.2.4), and ModelPicker (2.4.2). Among these components, DataMagic introduces uncertainties via weak supervision and automatic data cleaning, while other components eliminate uncertainties via manual data cleaning and labeling. One key design decision in Ease.ML is to build uncertainty elimination components with the same *principle* of information maximization. Directly applying this principle is often hard. We develop therefore different proxies and optimizations for each individual component. This is derived from the same underlying assumption that we call the GT-BEST Assumption: *the ground truth $G^*_{\mathbf{D}}$ always performs the best in terms of utility*. As a result, the process of maximizing the utility is to eliminate the uncertainties such that we are as close to the ground truth as possible.

Ease.ML/DataMagic. Given an input dataset provided by the user, DataMagic [25, 20] automatically parses it and generates a dataset that is consistent with the Ease.ML data model. Specifically, DataMagic introduces uncertainties with the following steps:

1. Run state-of-the-art data cleaning methods to induce candidate values for each feature vector and labels.

2. Conduct automatic data augmentation to induce candidate values for each feature vector.

3. If possible, run state-of-the-art weak supervision tools to induce uncertainties over labels.

After DataMagic, the system produces the four datasets: (1) a labeled training set $\mathbf{D}_{tr}$; (2) unlabeled data examples $\mathbf{D}_{unl}$; (3) a labeled validation set $\mathbf{D}_{val}$; and (4) an unlabeled production test set $\mathbf{D}_{prod}$.

Ease.ML/CPClean. The goal of Ease.ML/CPClean [11] is to eliminate uncertainties over features via manual data cleaning. It takes as input the validation set $\mathbf{D}_{val}$ and prioritizes the cleaning of examples in the training set $\mathbf{D}_{tr}$. The goal is to find a subset of K data examples $\pi \subseteq [|\mathbf{D}_{tr}|]$ to clean such that the expected validation accuracy is maximized. Within our framework, this is to solve

$$\max_{(\mathbf{x}_\pi, \mathbf{y}_\pi) \subseteq \mathbf{D}_{tr}; |\pi| = K} \mathcal{H}[\mathcal{A}_{\mathbf{D}_{tr}}(\mathbf{D}_{val}) \mid (\mathbf{x}_\pi, \mathbf{y}_\pi) = (\mathbf{x}^*_\pi, \mathbf{y}^*_\pi)].$$

Directly optimizing for this objective is hard. In CPClean, we provide an efficient PTIME algorithm to calculate the entropy $\mathcal{H}$ for a subfamily of classifiers $\mathcal{A}$ [11], namely K-nearest neighbor classifiers, and use it as the proxy for more advanced classifiers.

Ease.ML/Market. The goal of Ease.ML/Market [8, 6, 7] is to eliminate uncertainties over the labels of unlabeled examples via manual labeling. It takes as input the current training set $\mathbf{D}_{tr}$, the pool of unlabeled examples $\mathbf{D}_{unl}$, and the validation set $\mathbf{D}_{val}$. The goal is to pick K data examples to label in $\mathbf{D}_{unl}$ to maximize the expected validation accuracy. Within our framework, this is to solve

$$\max_{(\mathbf{x}_\pi, \mathbf{y}_\pi) \subseteq \mathbf{D}_{unl}; |\pi| = K} \mathcal{H}[\mathcal{A}_{\mathbf{D}_{tr} \cup \mathbf{D}_{unl}}(\mathbf{D}_{val}) \mid (\mathbf{x}_\pi, \mathbf{y}_\pi) = (\mathbf{x}^*_\pi, \mathbf{y}^*_\pi)]$$

In Market, one design decision that we made is to treat this problem as a prediction problem — use the Shapley value of each data example with the right label as the proxy of the above term, and train a predictive model to predict such a value given the feature vector and apply it to unlabeled examples [7]. The calculation of the Shapley value can be hard for the general classifier, and we provide a PTIME algorithm for a subfamily of classifiers $\mathcal{A}$ [6], namely the K-nearest neighbor classifier, and use it as the proxy.

Ease.ML/ModelPicker. Given a collection of candidate models $M_1 ... M_k$, the goal of Ease.ml/ModelPicker is to pick one that has the maximum test accuracy by requiring as few labels as possible. It takes as input the unlabeled production test set $\mathbf{D}_{prod}$ and selects K data examples to label in order to maximize the confidence of picking the best model. In our framework, it is to solve

$$\max_{(x^*_\pi, \mathbf{y}_\pi) \subseteq \mathbf{D}_{prod}, |\pi| = K} \mathcal{H}[\mathcal{M}(\mathbf{D}_{prod}) \mid \mathbf{y}_\pi = y^*_\pi]$$

where $\mathcal{M}(D)$ returns a $k \times k$ matrix with $\mathcal{M}(D)_{i,j}$ indicating whether model M_i performs better than M_j on D. We show that it is also possible to design an efficient algorithm to optimize for certain versions of the above problem [9].

3.3.2 Quality Control, Estimation, and Optimization

The second collection of Ease.ML components aims at estimating the quality of ML models or automatically optimizing them. These components include: (1) Ease.ML/Snoopy [24], an automatic feasibility study tool with a practical Bayes error estimator, taking advantage of available embedding hubs such as TensorFlow Hub, (2) Ease.ML/ci [22, 10, 23], an automatic management system of statistical powers for test sets that controls the information leakage from the test set to the user during the continuous testing and integration process, and (3) Ease.ML/AutoML [36, 15, 12], an AutoML system that efficiently maintains applications given new ML models by taking a multi-tenant view.

3.4 A Prototype Implementation

We built a prototype implementation of our system to showcase the key aspects of our design. One key design decision is to base our system with Jupyter notebooks as it not only represents a familiar and powerful platform to many data scientists but also provides naturally all interactions that Ease.ML has with its users.

The user interaction in Jupyter notebooks is conducted through "code cells" where the user types in usually short code snippets and then executes them, after which the output is presented immediately below the cell. We use the same approach and we extend it with slightly enriched output cells that can contain various UI items such as buttons, progress bars and interactive graphs. The user loads the data after which she can execute any sequence of data manipulation operations (e.g. input preprocessing, cleaning, filtering) intertwined with various Ease.ML operations (e.g. Ease.ML/Snoopy or Ease.ML/AutoML).

Each manipulation is tracked and stored in a lineage graph that is used to represent the user's ML workflow. This graph can then be optimized, materialized, or stored for later use. A lineage graph that is saved can also be executed at a later point, e.g. through a command line shell or as part of some CI/CD workflow. Furthermore, the user can ask the system to give suggestions for next steps given the current state of the lineage graph.

As mentioned earlier, the logical data model we assume for all ML datasets is the probabilistic relational model where each row is associated with a single data example (i.e. a sample from a data distribution). Each data example is furthermore associated with one or more possible candidate rows from which only one can be selected at any given time. Each column has a type that can be either a simple scalar value (e.g. numeric, Boolean, categorical, string, etc.) or a more complex type (e.g. document, tensor, etc.) This data model is implemented with the `Relation` class that wraps around the `DataFrame` class taken from the popular Pandas framework which we extend with the mentioned capabilities.

To manipulate the data stored in our data model, the user would run various available operators. These operators work in a staged manner whereby invoking an operator merely appends it to the lineage graph. The return value of the operator invocation is the updated lineage graph. At any point in the interactive ML process the user can choose to materialize the graph by calling the `eval()` function. This process involves performing various optimizations after which the resulting data view is shown in the output cell of the Jupyter notebook while at the same time being cached for future use. This approach allows us to treat ML workflows as first-class citizens that can be managed like any other data artefact, while at the same time preserving the interactive nature of Jupyter notebooks.

4. CONCLUSION AND FUTURE WORK

We have presented Ease.ML, a novel platform that aims for automated end-to-end lifecycle management of ML application development. The current iteration of Ease.ML includes an eight-step process that covers the entire development lifecycle, motivated by the pitfalls and confusions we found during interactions with our users. We have also presented the foundation of Ease.ML, in connection with probabilistic database theory and information theory. As a research prototype, the goal of Ease.ML is to "expose" limitations in its design instead of "hide" them. We briefly describe the most severe limitations of the current design. In our opinion, these limitations indicate exciting directions for future research.

Efficiency. Whenever a design decision is made, the current Ease.ML prioritizes more on a principled formulation rather than an efficient way. One example is the unified information maximization framework for all human-related interaction. Although for each of these tasks, we developed different proxy models and approximations, it can be beneficial to explore how to conduct each task directly on the target black-box training procedure.

Beyond Accuracy. The current design of the Ease.ML process overwhelmingly focuses on a single target metric, i.e., accuracy. This is a reflection of the focus of many ML systems and tools. Recently, there has been an emerging collection of metrics, such as robustness, fairness, and explainability, that has attracted intensive interest. How to systematically support the engineering process of these new metrics is an interesting direction, in our opinion.

5. ACKNOWLEDGEMENT

CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Oracle Labs, Swisscom, Zurich Insurance, Chinese Scholarship Council, and the Department of Computer Science at ETH Zurich.

6. REFERENCES

[1] Sandro Ackermann et al. Using transfer learning to detect galaxy mergers. *MNRAS*, 2018.

[2] Johannes Beck et al. Sensing social media signals for cryptocurrency news. In *Companion Proceedings of WWW 2019*, 2019.

[3] Eric Breck et al. Data validation for machine learning. In *SysML*, 2019.

[4] Ivan Girardi et al. Patient risk assessment and warning symptom detection using deep attention-based neural networks. In *The Ninth International Workshop on Health Text Mining and Information Analysis*, 2018.

[5] Nina Glaser et al. Radiogan–translations between different radio surveys with generative adversarial networks. *MNRAS*, 2019.

[6] Ruoxi Jia et al. Efficient task-specific data valuation for nearest neighbor algorithms. *PVLDB*, 2019.

[7] Ruoxi Jia et al. An empirical and comparative analysis of data valuation with scalable algorithms. *arXiv*, 2019.

[8] Ruoxi Jia et al. Towards efficient data valuation based on the shapley value. AISTATS, 2019.

[9] Mohammad Reza Karimi, Nezihe Merve Gürel, Bojan Karlaš, Johannes Rausch, Ce Zhang, and Andreas Krause. Online active model selection for pre-trained classifiers, 2020.

[10] Bojan Karlas et al. Building continuous integration services for machine learning. In *KDD*, 2020.

[11] Bojan Karlaš et al. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv*, 2020.

[12] Bojan Karlaš et al. Ease.ml in action: Towards multi-tenant declarative learning services. *VLDB Demo*, 2018.

[13] Tim Kraska. Northstar: An interactive data science system. *PVLDB*, 2018.

[14] Sanjay Krishnan et al. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 2016.

[15] Tian Li et al. Ease.ml: Towards multi-tenant resource sharing for machine learning workloads. In *PVLDB*, 2018.

[16] Akshay Naresh Modi et al. Tfx: A tensorflow-based production-scale machine learning platform. In *KDD 2017*, 2017.

[17] Supun Nakandala et al. Incremental and approximate inference for faster occlusion-based deep cnn explanations. In *SIGMOD*, 2019.

[18] Supun Nakandala et al. Cerebro: A data system for optimized deep learning model selection. *PVLDB*, 2020.

[19] Alexander Ratner et al. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 2017.

[20] Johannes Rausch et al. Docparser: Hierarchical structure parsing of document renderings. *AAAI*, 2021.

[21] Theodoros Rekatsinas et al. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 2017.

[22] Cedric Renggli et al. Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment. In *SysML*, 2019.

[23] Cedric Renggli et al. Ease.ml/ci and ease.ml/meter in action: Towards data management for statistical generalization. In *VLDB Demo*, 2019.

[24] Cedric Renggli et al. Ease.ml/snoopy in action: Towards automatic feasibility analysis for machine learning application development. In *VLDB Demo*, 2020.

[25] Giuseppe Russo et al. Control, generate, augment: A scalable framework for multi-attribute text generation. *arXiv*, 2020.

[26] Lia F Sartori et al. A model for agn variability on multiple time-scales. *MNRAS: Letters*, 2018.

[27] Lia F Sartori et al. A forward modeling approach to agn variability–method description and early applications. *The Astrophysical Journal*, 2019.

[28] Kevin Schawinski et al. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *MNRAS Letters*, 2017.

[29] Kevin Schawinski et al. Exploring galaxy evolution with generative models. *Astronomy & Astrophysics*, 2018.

[30] Dominic Stark et al. Psfgan: a generative adversarial network system for separating quasar point sources and host galaxy light. *MNRAS*, 2018.

[31] Min Su et al. Generative adversarial networks as a tool to recover structural information from cryo-electron microscopy data. *bioRxiv*, 2018.

[32] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. 2011.

[33] Manasi Vartak et al. Modeldb: A system for machine learning model management. In *HILDA*, 2016.

[34] Renzhi Wu et al. Zeroer: Entity resolution using zero labeled examples. In *SIGMOD*, 2020.

[35] Weiyuan Wu et al. Complaint-driven training data debugging for query 2.0. In *SIGMOD*, 2020.

[36] Chen Yu et al. Automl from service provider's perspective: Multi-device, multi-tenant model selection with gp-ei. AISTATS, 2019.

[37] M. Zaharia et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 2018.

[38] Ce Zhang et al. Deepdive: Declarative knowledge base construction. *Commun. ACM*, 2017.