ABSTRACT

Developing machine learning (ML) applications is similar to developing traditional software — it is often an iterative process in which developers navigate within a rich space of requirements, design decisions, implementations, empirical quality, and performance. In traditional software development, software engineering is the field of study which provides principled guidelines for this iterative process. However, as of today, the counterpart of “software engineering for ML” is largely missing — developers of ML applications are left with powerful tools (e.g., TensorFlow and PyTorch) but little guidance regarding the development lifecycle itself.

In this paper, we view the management of ML development lifecycles from a data management perspective. We demonstrate two closely related systems, ease.ml/ci and ease.ml/meter, that provide some “principled guidelines” for ML application development: ci is a continuous integration engine for ML models and meter is a “profiler” for controlling overfitting of ML models. Both systems focus on managing the “statistical generalization power” of datasets used for assessing the quality of ML applications, namely, the validation set and the test set. By demonstrating these two systems we hope to spawn further discussions within our community on building this new type of data management systems for statistical generalization.

1. INTRODUCTION

Recent years have witnessed the rapid adaption of machine learning (ML) techniques to a diverse range of real-world applications. Part of this trend is enabled by the recent research on scalable ML training (e.g., [10]) and AutoML (e.g., [9, 11]), which significantly improve the usability of ML systems. Training a single ML model becomes faster and more automatic, and the portfolio of ML developers extends beyond ML experts. However, new challenges of usability arise, as developing ML applications, similar to developing traditional software, is an iterative process: ML systems should not only support the development of a single model, but also the iterative process as a whole.

(Motivating Examples) This work is motivated by our experience of supporting developers, most of whom are not ML experts, in building a range of ML applications [4, 7, 14, 15, 16, 17] over the last three years. Specifically, we are motivated by the struggles that our users face even after they have access to an AutoML system [11] with a scalable and efficient training engine [18] — when training an ML model becomes easier, users start to seek systematic guidelines, which, if followed step by step, can lead to the right model. This shift of focus is not dissimilar to what software engineering tries to provide for traditional software development. Typical questions that our users frequently asked include:

(Q1) Is my new model really better than my old model?

(Q2) How large does my validation set need to be? Is my current validation set still representative?

(Q3) How large does my test set need to be? Is my current test set still representative?

(Data Management for Statistical Generalization) These questions are essentially asking about the statistical generalization (ability) of a data set. Let $M$ be a model, $D_{test}$ be a test set, and let $D_{test}$ be the (unknown) true underlying data distribution from which $D_{test}$ is drawn. The questions posed above can be answered (see Section 3) if the system has access to a representative test score $S(M, D_{test})$ that is similar to the “true” score $S(M, D_{true})$. The fundamental challenge is that the “statistical power” of $D_{test}$ may fade once it has been used to provide an answer (i.e., if $D_{test}$ starts to overfit). As a result, any system we build to answer these three questions must be able to automatically manage the statistical power of a given dataset. This leads to a new type of data management system which, instead of managing the (relational) querying of the data, manages the statistical generalization of the data.

(Summary of Demonstration Plans) In this proposal, we plan to demonstrate ease.ml/ci and ease.ml/meter, two closely related systems that we have built to manage the statistical power

---

The two first authors contributed equally to this work.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097
DOI: https://doi.org/10.14778/3352063.3352110
of a test set to facilitate developer’s understanding of the motivating questions from above. Specifically, we will focus on the following aspects in our demonstration:

1. A demonstration of multiple real-world use cases that motivate the development of both systems;
2. A demonstration of the overfitting problem if we do not take a data management view when building these systems;
3. A demonstration of the functionality of both systems using real development traces of multiple ML applications;
4. Interaction with the audience to test the system with syntethic development traces provided interactively;
5. Discussion with the audience about future research directions in data management for statistical generalization.

2. SYSTEM OVERVIEW

Building ease.ml/ci and ease.ml/meter is not trivial – the technical details are mainly developed and covered in our latest full research papers [8][12]. In this section we provide a brief overview of both systems.

Figure 1 illustrates the interfaces of both systems. One can think about the common abstraction of both systems as “GitHub for ML” — developers commit a stream of ML models and get feedback signals for each model committed. All data sets (e.g., training, validation, and test) are managed by the systems — developers do not have access to the test set, while labelers (who provide test sets) have full access to the test set. Apart from providing signals to developers, the systems also ask labelers for new labels if required.

We next describe the interaction model of ease.ml/ci and ease.ml/meter in more detail.

2.1 Ease.ml/ci: Functionality

Figure 2 illustrates how users interact with ease.ml/ci. Similar to traditional systems for continuous integration, ease.ml/ci allows developers to define test conditions that specify quality assertions of ML models. Examples of such assertions include:

(A) The new version \( z \) of the ML model must be at least 2% more accurate than the old version \( \phi \).

Unlike traditional test assertions that raise a binary “pass/fail” signal, quality assertions in ease.ml/ci are not deterministic, due to the statistical nature of quantifying the model’s power to generalize. Instead, we provide rigorous probabilistic guarantees for the

\[ \Pr \left( \left| \bar{O}(\phi) - \bar{O}(z) \right| \leq \epsilon \right) \geq 1 - \delta \]

Figure 3: User interaction with ease.ml/meter.

assertions. Developers can specify the error tolerance of a test condition following an \((\epsilon, \delta)-\)reliability paradigm: The test condition holds within the \(\epsilon\)-error bound with probability at least \(1 - \delta\).

ease.ml/ci supports various forms of assertions based on a novel scripting language that enables user interactions in a declarative manner. Figure 3 also presents an example script, specifying the same assertion (A) but with error bound \(\epsilon = 0.01\) and reliability (confidence) 0.9999 (i.e., \(\delta = 0.0001\)).

2.2 Ease.ml/meter: Functionality

Figure 4 illustrates user interactions with ease.ml/meter. A human developer starts from an initial ML application \(H_0\), and performs the following two steps in each development iteration \(i\):

1. Error Analysis. The developer looks at the current validation set and conducts error analysis on all errors currently being made by \(H_i\);

2. Development. The developer then makes changes on the current ML application \(H_i\), for example, by adding new features, adding more labeling functions, trying a different hyperparameter setting, etc., to produce a new version of the ML application \(H_{i+1}\).

Upon receiving the latest submission \(H_i\), ease.ml/meter evaluates its accuracy over the (hidden) test set and returns an overfitting signal to the developer indicating the degree of overfitting of \(H_i\). Specifically, ease.ml/meter employs an overfitting method for delivering overfitting signals to developers. The method consists of \(m\) overfitting intervals \([r_i, \bar{r}_i]\) with \(i\) that partitions the range \([0,1]\) (e.g., \(m = 5\) for the meter in Figure 3). Each range \(R_i = [r_i, \bar{r}_i]\) is further associated with an \(\epsilon_i\) (0 < \(\epsilon_i\) < 1) that indicates the error bound of the estimated overfitting. Similar to the probabilistic guarantee (i.e., the \((\epsilon, \delta)-\)reliability) provided by ease.ml/ci for test conditions, ease.ml/meter provides probabilistic guarantee for the overfitting signal it returns: The degree of overfitting is bounded by \([r_i - \epsilon_i, \bar{r}_i + \epsilon_i]\) with probability (confidence) at least \(1 - \delta\), if the meter returns signal \(i\) (1 ≤ \(i\) ≤ \(m\)).

We assume that in both systems developers do not “collude” with labelers, e.g., labelers cannot send a copy of the test set to developers without having the systems know it.
2.3 Core Technical Challenges and Solutions

The key (common) technical challenge faced by both systems, ease.ml/ci and ease.ml/meter, is to ensure probabilistic guarantees (for either test assertions in ease.ml/ci or overfitting signals in ease.ml/meter) via the $(\epsilon,\delta)$-reliability paradigm. A naive approach is to draw a new test set every time a new version of the ML application was submitted. Unfortunately, this “resampling” approach is often unfeasible in practice due to its reliance on a huge amount of labeled test data.

Specifically, to address the challenge in a generic setting, one has to deal with dependencies between subsequent submissions during the development lifecycle, due to the presence of adaptive analysis: The performance of the $i^\text{th}$ model submission $f_i(\{X_i\})$ on the test set $\{X_i\}$ is reported back to the developer indirectly via a feedback/reporting function $g(\cdot)$. The developer can then take an adaptive action by picking the next submission $f_{i+1}(\{X_i\})$ based on the value returned by $g(\cdot)$. As was shown by recent work [5, 6], such kind of adaptive analysis can accelerate overfitting on the test set. As a result, to maintain the same level of statistical generalization power, one needs a larger test set (compared to the case where all submissions are independent). The technical contribution of both systems is a collection of novel statistical estimators that decreases the required test set size by two orders of magnitude — this reduction is critical for the viability of both systems. For more details, see Renggli et al. [12].

3. DEMONSTRATION SCENARIOS

We present details of the scenarios that we plan to demonstrate with ease.ml/ci and ease.ml/meter. We will use real-world development traces of two ML applications that we developed over the last two years. Each trace contains eight sequential learning models from the ML application construction. Figure 4 illustrates the evolution of validation/test accuracy of these models.

Trace 1: Relation Extraction. The first example trace comes from our submission to the “Semantic Relation Extraction and Classification in Scientific Papers” task in SemEval 2018 [13]. It is the top-ranked system for three out of four subtasks. This task aims to identify concepts from scientific documents and recognize semantic relationships that hold between the concepts. In particular, it requires semantic relation extraction and classification into six categories specific to scientific literature. The development history involves eight intermediate models before reaching the final system. Figure 4(a) plots the accuracy on the training set (using 5-fold cross validation) and the test set, respectively, in each step.

Trace 2: Emotion Detection. The second example comes from the development history of our submission to the “EmoContext” task in SemEval 2019. This task aims to detect emotions from text, leveraging contextual information, which is deemed challenging due to the lack of facial expressions and voice modulations. It took developers eight iterations before delivering the final version. Changes in each individual step include adding word representations such as ELMo and GloVe, which lead to significant performance increase/drop. Figure 4(b) plots the accuracy on the validation set and the test set for in each step, respectively (assuming that the accuracy on the test set were reported to the user in every development step).

3.1 Scenario 1: Without-Data Management

We will start by highlighting the importance of data management in ML application development lifecycles. We plan to showcase a couple of scenarios, where developers, intentionally or not, overfitted to the test set. In particular, we focus on demonstrating the following aspects: (1) If there is no data management system and the developers have full access to the test set, overfitting on the test set occurs easily in continuous development; (2) Even if we control its direct access, overfitting on the test set may still occur easily if the developers have access to the test result. We hope that this part would convince the audience of the motivation behind ease.ml/ci and ease.ml/meter.

3.2 Scenario 2: Ease.ml/ci

We then showcase the functionality of ease.ml/ci by using trace data from real-world ML application development activities. Figure 5 presents some example traces.

The development history in Figure 5 involves eight individual steps and three different scripts expressing test assertions. The first two assertions test whether the new model is better than the previous one by at least 2% (in terms of accuracy) in a non-adaptive setting, i.e., by assuming the models are independent. fp-free and fn-free further represent different modes that ease.ml/ci is operating on, enforcing elimination of false positives and false negatives, respectively. Meanwhile, the third assertion further mimics the scenario where developers can get feedback in each commit without false negatives. All these three scripting queries can be supported (in a rigorous manner) with about 5.5K test samples.

3.3 Scenario 3: Ease.ml/meter

We then showcase the functionality of ease.ml/meter by using the same trace data. Figure 6 illustrates the signals that the developer would get during the development process, using different modes of ease.ml/meter. By putting together the quality characteristics of both traces (Figure 5 and the signals from ease.ml/meter (Figure 6), we will guide the audience through different instances of overfitting behavior and elaborate how the
There is a great deal of recent work on developing "declarative" ML services to alleviate development efforts. In a typical AutoML system, users only need to upload their data to the system, and the system will automatically create models and select the best one based on performance metrics. This allows users to focus on the data and not the technical details of model selection and tuning.

In this paper, we focus on the struggles that our users have even after they have access to these AutoML systems, and hope to provide more systematic (and automatic) guidelines to help users use these powerful AutoML systems better.

**Adaptive Analysis.** Recent theoretical work [6] has revealed how adaptive analysis increases the risk of false conclusions reached by statistical methods (known as "false discovery" in the literature).

Intuitively, adaptive analysis makes it more likely to draw conclusions tied to the specific data set used in a statistical study, rather than conclusions that can be generalized to the underlying distribution that governs data generation. Our work in this paper can be viewed as an application of this theoretical observation to a novel, important scenario emerging from the field of ML application development, lifecycle management, and quality control.

**5. CONCLUSION**

We have demonstrated the importance of using AutoML systems for managing statistical generalization of test data sets in continuous ML application development activities. We view our current work as a first step towards data management in lifecycle control and quality assurance of ML applications, an emerging yet promising area that has not been paid enough attention by the database community. We hope that our work offers some insights into this fertile ground and provides interesting scenarios that could inspire future research.

**Acknowledgements** CZ and the DS3Lab gratefully acknowledge the support from the Swiss National Science Foundation (Project Number 200021_184628).

**6. REFERENCES**


