Context: The Missing Piece in the Machine Learning Lifecycle

Rolando Garcia, Vikram Sreekanti, Neeraja Yadwadkar, Daniel Crankshaw, Joseph E. Gonzalez, Joseph M. Hellerstein
UC Berkeley
{rogarcia, vikrams, neerajay, crankshaw, jegonzal, hellerstein}@berkeley.edu

ABSTRACT

Machine learning models have become ubiquitous in modern applications. The ML Lifecycle describes a three-phase process used by data scientists and data engineers to develop, train, and serve models. Unfortunately, context around the data, code, people, and systems involved in these pipelines is not captured today. In this paper, we first discuss common pitfalls that missing context creates. Some examples where context is missing include tracking the relationships between code and data and capturing experimental processes over time. We then discuss techniques to address these challenges and briefly mention future work around designing and implementing systems in this space.

1. INTRODUCTION

Most modern applications—ranging from personal voice assistants to manufacturing services—rely on machine learning in some form. These applications rely on machine learning *models* to render predictions in response to a query. The development, training, and serving of machine learning models is the result of a process that we call the *Machine Learning Lifecycle*. This lifecycle has three phases (Figure 1): pipeline development, training, and inference.

The pipeline development phase is an iterative process in which data scientists transform and visualize data, explore various model designs, and experiment with many features. Note that the focus on model design often leads to the term "model development." However, the true product of *pipeline development* is a reusable pipeline that describes how to construct a model from a given dataset. This pipeline is then executed on a much larger, near-real time dataset to generate a production-ready model, and these trained models are in turn used to serve predictions for new inputs to the application.

The ML Lifecycle is data-intensive and spans many individuals. Each stage is often managed by a different team, with different incentives and different structures. The transitions between stages and teams are usually *ad-hoc* and unstructured. As a result, in serious machine learning deployments today, no one person or system has an end-to-end view of the ML Lifecycle. This is problematic for a variety of reasons, including irreproducibility of experimental results, complicated debugging, and a lack of accountability. We believe there is a key missing component required to capture this view: the *context* that surrounds the ML Lifecycle.

Recent work highlights data context [5] as a critical aspect of any data-centric effort, including ML pipelines. That work defines data context as "all the information surrounding the use of data in an organization." It goes on to distinguish three key types of data context: application context, behavioral context, and change over time (the "ABCs" of context). The application context (A) captures semantic and statistical models that explain how bits should be interpreted. Behavioral context (B) extends the traditional notion of data provenance to capture how both people and software interact with data. Lastly, change over time (C) captures how each of the other two types of data context are evolving.

In the next section, we highlight common pitfalls that arise from ignoring data context in the ML Lifecycle. We then discuss approaches to contextualize the ML Lifecycle and briefly mention future work.

CMI '18 August 20, 2018, London, UK
© 2018 ACM.

2. THE ABSENCE OF CONTEXT

In this section, we describe how code and data should be interpreted, how they evolve over time, and what their relationship is to each other and the people that create and use them. Our discussion also includes the history of an organization's model management practices. We consider more than just the success stories: we consider the context surrounding the experiments that failed because of bugs, poor performance, excessive cost, and so on. We strive to leverage this context to learn from our mistakes and the mistakes of others, to reduce work duplication, and to formalize machine learning practices.

2.1 The Code and Data Ecosystem

Data is a first class citizen in any machine learning pipeline. The same pipeline trained on different data can yield a drastically different result. Unfortunately, most organizations today do not capture the relationship between code and data, both within and across the phases of the ML Lifecycle. Most importantly, context around which data set was used to train a particular version of a model is lost. Other basic information—like schemas, distributions, and expectations—are even less available.

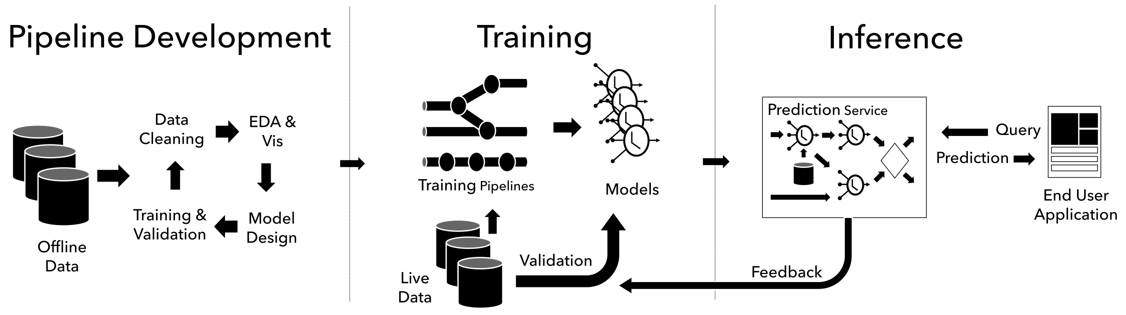

Figure 1: The end-to-end machine learning lifecycle.

In the rest of this section, we describe a few common scenarios demonstrating the loss of context in the code and data ecosystem.

This data looks wrong! The first step in the ML Lifecycle often consists of transforming raw data into a cleaned dataset. That dataset is often shared and reused. If a data scientist or analyst who receives the data encounter issues, they need access to the original data and transformation scripts. If this context is not explicitly logged, the derivation of the cleaned dataset is opaque to the receiving user. What's worse is that they may be unaware that the data they received was derived through a transformation process. **Missing Context:** the code and data used to construct the dataset.

If I could only find that model from last week! Pipeline development is inherently experimental with many iterations of trial and error. There are a variety of reasons that we may want to return to earlier versions of our models and data. For example, it is common to reach a dead-end in model design only to return to an earlier model. Recreating an earlier state requires reverting not only code but also data, parameters, and configurations. Finding the earlier best version may require searching through many alternative versions. **Missing Context:** the versions of code, data, parameters, and configurations over time.

It worked better yesterday! Models inevitably degrade in their predictive power. There are many reasons for this degradation. For example, a shift in the data distribution can result in a rapid decline in predictive power. Diagnosing this decline requires comparing training data with live data. Solving the problem may require retraining the model, revisiting earlier design decisions, or even rearchitecting the underlying model. **Missing Context:** the full lineage of the model through each stage of the ML Lifecycle as it existed at the time of training.

I fixed it, who needs to know? Models are routinely composed in production. For example, modeling a users likelihood to buy a product might depend on predictions about the user's political preference. Changes to upstream model will impact the quality of predictions from a downstream model. Surprisingly, improving the accuracy of an upstream model can in some cases degrade the accuracy of downstream models because the downstream models were trained to compensate for errors. **Missing Context:** the eventual consumers of predictions from models.

2.2 Learning From Your Mistakes

An analyst's first question when debugging might be to ask if they are encountering a well-known problem. Answering this requires context in two scopes. First, it requires context around past experiments for this pipeline—both successful and unsuccessful. It could also potentially require surrounding other, related projects in the organization. We next look at two examples of this sort, one diagnosing a problem and the other remediating a problem.

I've seen this problem before. Any organization will develop a set of recurring problems in their ML Lifecycle. Imagine a model for a ride-sharing service is predicting negative trip durations. A natural question might be to ask why the training data would lead to this prediction and whether there's a standard data cleaning step that is missing. An expert on trip data might know that canceled trips are logged as having -1 duration and that these trips should be dropped; however, this information is not readily available without experience. **Missing context:** Standard transformations applied to a dataset.

I tried that already! Imagine a data scientist is tasked with improving the performance of a certain predictor developed by a *different* team member. It is likely that the same set of common model designs has already been considered. Knowing this information can help avoid redundant work. **Missing context:** Past designs and their resulting test scores.

2.3 Proper Methodology

The statistical nature of the ML Lifecycle requires experimental discipline. To some degree, this discipline can be captured and checked by recording the behavior of analysts and the structure of the pipelines themselves. While no single approach will solve all of the problems surrounding statistical and methodological practices, context will play a crucial role.

Have I used this test data before? When developing models, it's important to separate training and testing datasets. Overuse of testing data during training can lead to poor generalization and performance. For example, tuning a parameter by repeatedly testing on the same data can lead to over-fitting. **Missing context:** Behavior of the analyst and content & structure of the pipeline.

3. SYSTEM REQUIREMENTS FOR MODEL LIFECYCLE MANAGEMENT

Software engineering systems exist for building complex software projects, managing versions, automatically testing and deploying built binaries, logging, monitoring, and so on. More importantly, software engineering systems may have close analogs for the ML Lifecycle, and could serve as a sound starting point for research. However, there are characteristics of the ML Lifecycle that are unique to model management, and as a result, we do not expect to find strong parallels from software engineering. The characteristics that are special to the ML Lifecycle are the following: unlike the software engineering lifecycle, the ML Lifecycle is *empirical*, *combinatorial*, and *data driven*. The ML Lifecycle is *empirical* and *combinatorial* because even the most experienced and meticulous pipeline engineers will have only a vague idea of the elements and structure of the final pipeline. Context will play a central role in navigating the vast space of possibilities: it will be necessary to understand which data artifacts were used to train which models, and with what configurations. The ML Lifecycle is also *data-driven* because the model, the output of training, is inextricably linked to the data it was trained on. Contrast this with a software project, where the build or compilation of such systems are data independent. Context will be indispensable for the data and sample management systems that are used throughout the phases of the ML Lifecycle. Finally, as [8] notes, the use of machine learning models in an application often results in substantial technical costs stemming from the failure of traditional abstractions and software design principles in the presence of machine learning. While decades of research in software engineering has established techniques and tools to manage the development and deployment of complex software applications, there has been very little research into managing the development and deployment of machine learning applications.

3.1 *Pipeline Development* Tools

For *pipeline development*, we must build tools with which we can easily change the elements and structure of pipelines and feel confident in undoing unsuccessful attempts. Tools that support and enable *pipeline development* are "composition tools" [7]. These tools must support hypothesis formation, evaluation of alternatives, interpretation of intermediary results, and dissemination of results. Experienced pipeline engineers tasked with designing an ML pipeline often have a vague idea of the elements and structure of the pipeline in advance, but the particular configuration and final architecture must be discovered. For example, the pipeline engineer may know to use a neural network rather than a Naive Bayes Classifier, but the final number of layers and neurons per layer are unknown at the onset. To the extent that *pipeline development* is a creative and empirical endeavor, pipeline engineers must be encouraged to explore the space of possible alternatives. Tools that encourage exploration must have "low viscosity", meaning that they should easily enable changes to all aspects of the pipeline [7]; additionally, these tools must have very good *undo* capabilities, so the pipeline engineer feels comfortable trying new things, and powerful yet efficient previewing mechanisms to limit the consumption of scarce and valuable resources (such as computing time, memory, and data) without impeding exploration.

There are many possible roles for context in *pipeline development* tools. As a more general form of meta-data, context can help us interpret, and therefore compare the artifacts within pipelines and across their versions. Consider how a version control system, such as Git, would benefit from context. Git, which is tailored for source control and the software engineering lifecycle, defines change semantics as line-by-line differences. But these semantics are meaningless for binary and data artifacts. To detect meaningful change in data, we should look at metadata properties including the schema, distributions across different attributes, or topics in the data, instead of the exact contents of the records. When comparing binary objects such as two different models, it will be much more useful to compare the metadata of these models – e.g., their accuracy, recall, training hyper-parameter configurations, and so on – rather than just *diffing* their binaries.

3.2 *Training* Systems

Frameworks such as TensorFlow manage many of the problems of distributed training at scale; namely, scheduling computation to run on different devices, and managing the communication between these devices [1]. However, data engineers are still responsible for managing very large datasets and their versions over time, provisioning resources, and controlling for variability inherent in training in the cloud (attributable to multi-tenancy, hardware, workload, and data variability, and so on). In many cases, as deployed models interact with the world they produce new data that can be used to retrain existing models. In these cases, automated systems will periodically re-train existing models in response to changes in data or code. Unlike *pipeline development*, *training* does not require any human design considerations, and the search space is exhaustively enumerable in principle. However, data and model management requirements increase substantially. Of special significance to *training* will be techniques to automatically decide when to train, leverage knowledge of multiple pipelines to improve training performance, and study mechanisms to mitigate the risk of over-fitting.

Now, we consider how training systems may leverage and benefit from context. Context may benefit training in two ways: mitigating the risk of over-fitting and surfacing opportunities for optimization. Context about the training data and the processes that generated it, as well as context about how and how often that data is used can help reduce the risk of over-fitting to the data. As for optimization opportunities, organizations or cloud service providers will simultaneously run many training jobs often. In this case, it would be extremely valuable if the pipeline training system could characterize the pipelines that it is running and compare them to find common or equivalent transformations or subgraphs. Today, this kind of context is not widely available to the distributed systems that train models. This means that the system cannot intelligently schedule similar pipelines to run together, and re-use or share intermediate artifacts. The failure to understand the resource needs (meta-data) of each action in the pipeline can also lead to lost opportunities to schedule the execution of dis-similar pipelines more optimally. In summary, missed application context about the pipelines results in re-computation and poor schedules.

3.3 *Inference* Systems

During *inference*, trained models are used to render predictions for new inputs. The primary systems challenge of inference is delivering low latency, highly available predictions under heavy and often bursty query load. However, an often overlooked but critical component of inference is managing model versions and tracking variation in queries and prediction errors. Understanding, how models are performing production and debugging their failures depends critically on capturing their provenance.

Inference is the ML Lifecycle phase that already takes the most advantage of context. Prediction serving systems monitor the end-user application for feedback to measure the quality of the predictions. Other ways in which context could help inference is by leveraging context describing the input and output interfaces of models, together with the metadata about the training data and intended use for the model. If this context is available, the prediction serving system can more intelligently compose or combine models or their predictions to improve robustness and decrease latency.

4. RELATED WORK

Data Context Services. Modern data analytics ecosystems can benefit from a rich notion of *data context*, which captures all the information surrounding the use of data in organization. Data context includes traditional notions of metadata combined with data provenance and version histories. Recent work has discussed systems, such as Ground [5] and ProvDB [6], that enable users to capture richer data context. As discussed earlier, our work here builds on data context. However, this paper's key contribution is a domain-specific discussion of the benefits of data context in one domain. In other words, the ideas discussed here consistute an "Aboveground" application discussed in [5].

Model Management & Serving. A variety of recent work [4, 2, 3] has focused on the *inference* stage of the end-to-end ML lifecycle shown in Figure 1. Data context is essential for prediction serving systems that are fundamentally disconnected from the pipeline development and training stages of the ML lifecycle. For instance, explaining change in prediction accuracy is challenging without access to full lineage of the models, training data, and any hyperparameters. Similarly, ModelDB [9] captures context in the *pipeline development* phase but is disconnected from the broader ML Lifecycle.

5. CONCLUSION

In this paper, we characterized the ML Lifecycle and postulated that the crucial missing piece within and across every phase is *context*: "all the information surrounding the use of data in an organization." The transitions between stages and teams are usually *ad-hoc* and unstructured, meaning no single individual or system will have a global, end-to-end view of the ML Lifecycle. This can lead to problems like irreproducibility, over-fitting, and missed opportunities for improved productivity, performance, and robustness.

Some of the scenarios that illustrate the problems of missing context include those where the code and data used to clean data are lost, past versions are irretrievable, and deployed models degrade in performance over time. We also consider organizational context, often termed "tribal knowledge", that can be leveraged to reduce work duplication and respond to common problems, and some of the behavioral context that is generated during a pipeline engineer's activities, and how this context can be used to characterize and potentially help them improve their process.

The ML Lifecycle has some similarities with the software engineering lifecycle, such as how engineers describe pipelines for building a binary artifact, how there is a need to maintain different versions over time, and so on. This observation motivates our recommendation of drawing inspiration and guidance from software engineering when designing the tools for ML Lifecycle. However, we also note that the ML Lifecycle has some important differences from software engineering: namely, the ML Lifecycle is *empirical*, *combinatorial*, and *data-driven*. We also argue that context will be the key component in supplementing existing tools and creating future ones for the service of the ML Lifecycle.

Finally, we would like to end by noting that we are actively developing a system called Flor[1], a context-first tool for managing the ML Lifecycle. Our initial focus has been on tooling the *pipeline development* process, but we hope that these techniques will be applicable in the broader scope of the ML Lifecycle.

6. REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 173–182, New York, NY, USA, 2014. ACM.

[3] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The missing piece in complex analytics: Low latency, scalable model management and serving with velox. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[4] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, 2017. USENIX Association.

[5] J. M. Hellerstein, V. Sreekanti, J. E. Gonzalez, J. Dalton, A. Dey, S. Nag, K. Ramachandran, S. Arora, A. Bhattacharyya, S. Das, et al. Ground: A data context service. In *CIDR*, 2017.

[6] H. Miao, A. Chavan, and A. Deshpande. Provdb: A system for lifecycle management of collaborative analysis workflows. *CoRR*, abs/1610.04963, 2016.

[7] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg. Design principles for tools to support creative thinking. 2005.

[8] D. Sculley, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. Machine learning: The high-interest credit card of technical debt. 2014.

[9] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: A system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 14:1–14:3, New York, NY, USA, 2016. ACM.

[1]https://github.com/ucbrise/flor