Common Problems with Creating Machine Learning Pipelines from Existing Code

Katie O'Leary, Makoto Uchida
Google, Inc.
Mountain View, CA, USA
{katieole, muchida} @google.com

ABSTRACT

We worked with over 100 participants in industry on developing machine learning (ML) pipelines. Working alongside ML platform owners, software engineers, DevOps engineers, and data scientists across industries, we migrated existing ML projects into ones with ML pipelines software systems, Kubeflow Pipelines (KFP) and TensorFlow Extended (TFX). In this workshop paper, we share common problems we observed when migrating existing ML code to an ML pipeline system.

CCS CONCEPTS

• Human-centered computing~Empirical studies in HCI
• Software and its engineering~Software creation and management • Computing methodologies~Machine learning

KEYWORDS

Machine learning, workflow orchestration

ACM Reference format:

Katie O'Leary, Makoto Uchida. 2020. Common Problems with Creating Machine Learning Pipelines from Existing Code. In *Workshop on MLOps Systems, Third Conference on Machine Learning and Systems (MLSys).*

1 Introduction

Machine learning pipelines are becoming essential for enterprise-grade ML systems because they accomplish: 1) automated orchestration of workflow steps for model training and predictions, 2) reproducibility and tracking of executions and produced outputs of an ML workflow, and, 3) reusability of common workflow steps across multiple ML application systems. Enterprises are adopting pipelines products such as Apache Airflow[1], TensorFlow Extended (TFX)[2], and Kubeflow Pipelines (KFP)[3], to productionize their ML models. However, we are still in the early days of production ML and the challenges of implementing ML pipeline systems can be as high as the rewards, as evidenced by Spotify's recent blog post [1]. We contribute insights from observing ML teams migrate ML coding projects to pipelines systems, specifically TFX and KFP (both open-source projects), and some implications for design and research.

2 Method

We engaged over 100 participants across several industries transforming their business through machine learning. Our team of software engineers and UX researchers met with enterprise ML teams in a coding workshop setting, with the goal to implement ML pipelines for their own business applications. We involved their cross-functional ML teams of CEOs, DevOps, data scientists, VPs, and engineers. To preserve confidentiality, this paper synthesizes learnings without focusing on any single industry or team.

3 Common problems

3.1 The ML code 'dead end'

Data scientists often develop ML models in an iterative manner, typically in a local environment (such as notebook) with a snapshot of an offline dataset. The goal is to define and achieve an optimal model that satisfies business requirements. Due to the highly iterative nature of model development, the code is typically not authored in a robust software engineering program, nor does it have to be at this point [2]. However, this model-centric workflow of data scientists becomes problematic for software and DevOps engineers who operate the model in production, but must uphold the quality of the software being deployed. To deploy ML models into production, it must be integrated with the data infrastructure that would produce live training data, as well as with the serving

[1] https://airflow.apache.org/

[2] https://www.tensorflow.org/tfx

[3] https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/

infrastructure for the end-user application, along with monitoring, validation, data streams, etc., as described in [3]. In particular, online serving (scoring) systems with ML models often pose their own requirements that may not be obvious at modeling time, such as latency and model freshness constraints. As a result, engineers often need to re-implement the model from scratch into a deployable software. During the re-implementation, many of the implicit assumptions and nuances made by data scientists for modeling can be lost, resulting in unexpected inconsistencies and issues in production.

Such 'dead end' happens because data scientists compose models, or at best a piece of code to produce models, but not pipelines. Shifting to a pipeline-centric workflow is challenging for data scientists, because it requires them to formalize the programs before and after model training in ways that can be properly unit tested and deployed. Even if they do, model development with a formal pipeline program is slower than with an iterative environment such as notebook, due to the overhead of added software abstractions. We found that for enterprise ML teams, the conceptual leap from an ML model—which in fact is a collection of weights and variables rather than a piece of code—to a *pipeline system* was difficult, and posed a major hurdle to adoption.

3.2 Monolithic program

Data scientists tend to develop models with a monolithic program, *i.e.,* it may not be factored into functions and constructs of logical units as is a best practice in software engineering. A major value proposition of ML orchestration systems is to provide a framework to define individual workflow steps as *components* so that the data produced by each step can be formalized, and steps can be reused across different pipelines to reduce inefficiency. The concept of *components* is therefore essential to successfully adopting ML pipelines systems, and yet components are not necessarily intuitive to design. To design components, teams must decide on how to break down monolithic program code into canonical workflow steps, and then define clear interfaces for data passing between them. These challenges can be a barrier to migrating existing code to pipeline systems, as evidenced by teams who built a single, monolithic component "train," as their first pipeline, which in fact contains much more than model fitting code such as data loading, data transformation, and evaluation. Enterprise teams expressed difficulty in understanding how to design logical, canonical, and reusable components for pipelines.

3.3 Leveraging premade components

Finally, once ML teams (1) understand producing *pipelines* as the end deliverable of model development (versus a model); and (2) break the code into logical workflow steps, *i.e., components*, they want to (3) leverage and reuse premade 'authoritative' components that implement best practices of production ML [4], as opposed to reinventing the wheel from scratch. We observed that Kubeflow Pipelines, a highly flexible ML pipelines system, accommodated diverse ML topologies, yet imposed high user burden to design technically sophisticated components for common ML steps. Teams expressed interest in ready-made components, including templates, that could help them to adopt and customize ML-specific workflow steps for tasks such as automated model analysis and validation. TensorFlow Extended provides such components for best practices [5]. However, it also requires that other custom components in the pipeline be implemented in particular ways to match with out-of-box components. Given (1) and (2) are already non-trivial, refactoring the code to implement custom components to interoperate with out-of-box components implemented elsewhere, poses a further challenge.

CONCLUSION

Many organizations have well established practices for developing ML models, but we have observed a substantial—and as yet unsupported—conceptual leap in migrating from monolithic ML programs to componentized ML pipelines. This conceptual leap makes it difficult to use and adopt ML pipelines systems and poses barriers to productionizing machine learning at scale. To smooth out the enterprise journey to production ML systems, we have identified the following opportunities: 1) The environment for prototyping ML models should be designed to prevent the need to re-implement from scratch for production, 2) ML pipelines should provide a framework of pre-defined canonical unit of operations as components such that ML code can follow ML engineering best practices [6], as opposed to free-form flexibility, 3) Interfaces between components—both code and data—should be made explicit and simple enough so that implementing such interface is easy to use for ML code authors.

ACKNOWLEDGMENTS

For facilitating customer workshops, we thank: Jennifer Otitigbe, Sina Chavoshi, Hongye Sun, Alexey Volkov, Julian Atkinson, Hallie Crosby, Lucio Floretta, Erwin Huizenga, Khalid Salama, Amy Unruh, Scott Fitzharris, and Alex Walker. For pioneering discussions on emerging paradigm of ML Engineering, we thank: Robert Crowe, Konstantinos Katsiapis, Anusha Ramesh, Ben Mathes, Tris Warkentin, Salem Haykal, and Anand Iyer.

REFERENCES

[1] Josh Baer and Samuel Ngahane. 2019. The Winding Road to Better Machine Learning Infrastructure Through Tensorflow Extended and Kubeflow. Spotify Labs blog https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/

[2] Adam Rule, Aurélien Tabard, James Hollan. Exploration and Explanation in Computational Notebooks. ACM CHI Conference on Human Factors in Computing Systems, Apr 2018, Montréal, Canada. pp.1-12, ff10.1145/3173574.3173606.

[3] D Sculley, Gary Holt, Daniel Golovin, et al. 2015. Hidden Technical Debt in Machine Learning Systems. In C. Cortes, N.D. Lawrence, D.D. Lee, M.

Sugiyama, and R. Garnett, eds., Advances in Neural Information Processing Systems 28. Curran Associates, Inc., 2503–2511.

[4] Martin Zinkevich. 2018. Rules of Machine Learning: Best Practices for ML Engineering. https://developers.google.com/machine-learning/guides/rules-of-ml

[5] Denis Baylor, Eric Breck, Heng-Tze Cheng, et al. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM Press, New York, NY, 1387–1395. DOI:https://doi.org/10.1145/3097983.3098021

[6] Konstantinos Katsiapis and Kevin Haas. 2019. Towards ML Engineering with TensorFlow Extended (TFX). In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19*). ACM, New York, NY, 3182. DOI:https://doi.org/10.1145/3292500.3340408.