Machine Learning Systems and Intelligent Applications

William C. Benton
willb@redhat.com

***Abstract*—Artificial intelligence is enjoying an extended renaissance due to numerous successes of machine learning in many application areas. However, developing effective machine learning *techniques* is insufficient: one must also build and maintain machine learning *systems* that use these techniques in order to solve real business problems. These machine learning systems are necessarily complex and present myriad engineering challenges. We introduce the *intelligent applications* concept, which characterizes the structure and responsibilities of contemporary machine learning systems. Finally, we argue that Kubernetes is well-suited to taming the complexity of machine learning systems for the same reasons it has tamed the complexity of conventional distributed applications: declarative deployments, improved observability, and a single management interface for all application components.**

***Index Terms*—Artificial Intelligence, machine learning, distributed applications.**

I. Introduction

In order to solve a problem with machine learning, practitioners engage in several *discovery tasks*: they must *formalize a measure of success*; they must *identify, clean, and label a set of training examples*; they must devise a technique for *extracting structure from the data*; they must *train a model to exploit that structure*; and they must *validate that the model has generalized beyond its training data* by testing its performance on novel data. The output of this human discovery process is a trained model, but it is also the artifacts of the intermediate steps: a collection of cleaned example data, some techniques for turning this data into machine-readable features, and a technique for turning machine-readable features into a trained model. The discovery process is often realized manually as a collection of ad-hoc transformations, which harms repeatability.

In some cases, "production" is trivial since the model is never actually executed in production. Insights from models might be incorporated into real systems – for example, arranging a physical retail store so that items that are commonly purchased together occupy nearby shelves. However, machine learning is increasingly used to support essential functionality in software systems, like supporting commerce with personalized recommendations, evaluating payments transactions for probable fraud, or automatically managing a portfolio of securities.

In order to support the dynamic demands of software systems, we must not only publish models as production services but we must also build services to reproduce the data pipeline, feature extraction approach, and model training processes. Ideally the training steps will be deployed as a production pipeline – which takes cleaned data, extracts features, and trains a model – and the model itself will be deployed as a scoring pipeline, which takes raw data, extracts features, and makes predictions. Once the model is in production, the performance of the whole system must be continuously monitored – since production pipelines are often distributed systems and machine learning models are black boxes, their behavior can fail in many places and in nonobvious ways.

Figure 1 illustrates these discovery and production phases of a typical practitioner's workflow, showing back edges where a practitioner may need to revisit an earlier task. Since this workflow is necessarily iterative and can involve orchestrating numerous steps in a controlled environment, it can benefit from automation. In both respects, the machine learning workflow is analogous to a conventional software development workflow, and the same sorts of tools and infrastructure that have made it possible for developers to build and maintain increasingly complicated conventional applications can be adapted to serve the requirements of machine learning workflows and systems.

This paper will review the challenges involved in building and maintaining machine learning systems, introduce the *intelligent applications* concept as a model for contemporary machine learning systems, and show how open-source community projects

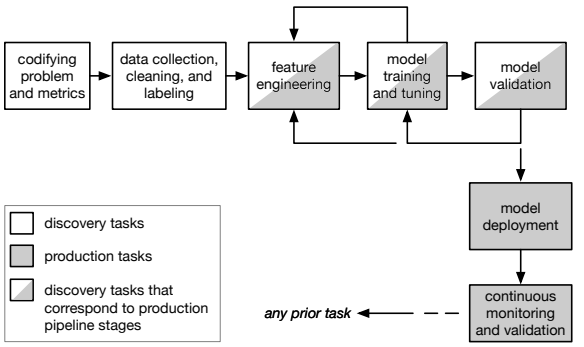

Fig. 1. A typical machine learning practitioner's workflow, showing both discovery and production tasks.

(including the Kubernetes resource manager and its ecosystem) can and do support intelligent applications and machine learning workloads in practice.

II. Machine learning systems

Machine learning techniques – that is, the feature engineering approach and feature extraction code, the actual optimization code that trains a model, and the inference code that uses it to make predictions – are important, but it is complex software systems that solve real problems. Machine learning systems incorporate, at a minimum, both training and inference pipelines and are thus multi-component and (often-)distributed systems that must deal with processed data from many sources and raw data from potentially-uncooperative users. As a consequence, machine learning systems feature many engineering challenges that are familiar to developers of complex applications, including:

- modules must be tested in isolation and the system must also be tested as a whole;
- application components must be built from source code, deployed, and orchestrated;
- it must be possible to upgrade individual components of the system without downtime;
- data formats, schemas, and ranges may change in ways that may violate the expectations of the code that consumes it; and
- the overall system must behave appropriately even in the face of hostile input.

Some challenges faced by conventional applications are more pronounced in machine learning systems: for example, configuring and orchestrating experiments and training pipelines is analogous to configuring and orchestrating conventional build pipelines, but the impact of a machine learning pipeline's configuration can be dramatic on the overall performance of a machine learning system. Making pipelines repeatable is an important goal: Nelson et al. (2011) describe the OURMINE environment for documenting and repeatably orchestrating polyglot pipelines, Buitinck et al. (2013) describe the `pipelines` API provided by Scikit-Learn (Pedregosa et al., 2011), which is suitable for pipelines developed in the Python language, and the Apache Spark system (Zaharia et al., 2016) provides an API for defining repeatable pipelines with relational queries, feature extraction, and model training steps.

However, other challenges are more insidious. For example, while conventional software often fails in obvious ways, machine learning systems may fail more subtly: a misbehaving model, for example, may continue to happily make predictions, but these will be wrong more than we'd like. Sculley et al. (2015) argue that *machine learning techniques* are relatively easy to develop but that *machine learning systems* are relatively difficult to maintain, and that machine learning systems often exhibit bad engineering properties due to the consequences of new challenges like:

- *concept drift* and other similar phenomena, in which relationships between inputs and outputs from training no longer hold in production, leading to decreased model performance;[1]
- unexpected changes to input *data quality* or pipeline correctness may have far-reaching effects in systems that depend on the quality of input data for the quality of trained models; and
- developing, managing, and monitoring the *integration code* that connects different components is tedious and error-prone.

III. Intelligent applications

Machine learning techniques, like business analytics and classical statistics, can provide value even without being incorporated directly into a software system: a monthly demand forecast can help a hospital schedule staff or a quarterly analysis of an organization's customers may identify those who are churn risks and should receive sales calls. In these kinds of applications, the output of a machine learning technique is an input to another application, and machine learning is thus a separate workload (often logically and physically) from application workloads.

[1]See Gama et al. (2014) for a recent survey of concept drift.

In other applications, models provide auxiliary functionality (e.g., printing personalized coupons based on a model of a retail customer's behavior) and are periodically trained, operationalized, and deployed into an application along with services to clean raw data, extract features, and make a prediction. In these applications, machine learning is again a separate workload, although the scoring pipeline may be ultimately incorporated into an application.

Intelligent applications also use machine learning to solve problems and derive business value. However, intelligent applications are different from these other approaches in several ways:

1) Intelligent applications are different in *what they use machine learning for*: intelligent applications employ machine learning models to support *essential* functionality and are thus able to provide improved performance with longevity and popularity.
2) Intelligent applications are different in *how they are developed*, namely, by cross-functional teams including data engineers (who make data available at scale), machine learning practitioners (who develop techniques to exploit patterns in data), and conventional developers (who build systems that depend on the data and machine learning techniques).
3) Finally, and most importantly, intelligent applications are different in *how they are deployed*: because it is essential to intelligent applications, machine learning *is not a separate workload* for them. Many of the challenges of machine learning systems are consequences of introducing opaque machine learning models and brittle glue code into a distributed application that would be difficult to manage even without machine learning; we can ameliorate these challenges to some extent if it is possible to manage all of the components of intelligent applications in a single control plane.

Each of these differences has consequences for how intelligent applications are designed, how they are built, and how they are deployed. Since machine learning supports essential application functionality, intelligent applications *must* continuously monitor data quality and model performance, retraining models when necessary. (An application in which a model provides periodic inputs to an application or in which a model supports auxiliary functionality can monitor model performance less regularly.) Since intelligent applications are developed by cross-functional teams, application infrastructure that supports collaboration is important. Since intelligent application development involves a traditional software development workflow and a machine learning workflow (like the one of Figure 1) in parallel, it also requires application infrastructure that can support both. Finally, since many of the challenges of machine learning systems are challenges of integration, management, and monitoring, intelligent applications benefit from mature infrastructure for distributed systems.

Architecture and responsibilities

An intelligent application *federates data from multiple sources*, including data in motion, structured data at rest (e.g., in databases), and unstructured data at rest (e.g., in file or object storage).

These data are then *processed, transformed, and cleaned*. Data for which ground truth is available are *labeled* to serve as part of a training set, which is further processed to *extract features* and passed to a model training algorithm, which deploys a standalone model service.[2] Raw data, which the application must act on, are similarly cleaned and processed before being scored by a trained model; all cleaned data are saved to archival storage (yesterday's raw data plus today's ground truth can be tomorrow's training data).

Conventional application components, like end-user interfaces, consume data and predictions and use traditional business logic to act on them. Every application component and data source produces metrics, which drive reporting dashboards and automated alerts (when it is possible to detect application crashes, concept drift, or data quality issues). Finally, a specialized management interface allows operators to scale and manage the application and a development interface allows data scientists to experiment in a replica of the live application environment or debug problems in production. Figure 2 depicts this architecture.

Intelligent applications are interesting *commercially* because most of today's most compelling and lucrative applications rely on machine learning to provide essential functionality in a complex system;*socially* because the lifecycles of these systems involve a cross-functional team with a range of skillsets; and

[2]For some kinds of models, it may make more sense to publish model parameters to stable storage or an in-memory cache for another service to load.

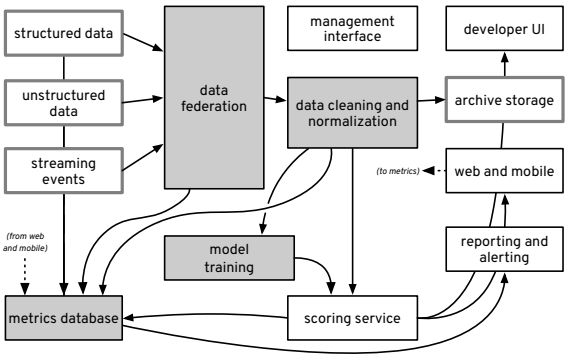

Fig. 2. An idealized intelligent-application architecture.

technically because machine learning systems present many engineering challenges.

The commercial aspect implies that developer velocity is at a premium while developing intelligent applications. The social aspect involves effectively building a cohesive system that depends on the talents of a diverse team even though software systems typically reflect the structure of the teams that build them (Conway, 1968). The technical aspect subsists in the challenge of managing, monitoring, and isolating a complex system with at least one opaque box in the middle – since there are so many places in which machine learning systems can go wrong, we need a single place to manage and observe the behavior of every component and their interactions.

IV. Kubernetes and intelligent applications

Sculley et al. (2015) showed that machine learning systems present new challenges but, paradoxically, that much of the engineering effort involved in developing and maintaining machine learning systems is not really specific to machine learning. Issues like data integration, process scheduling, configuration management, and resource management apply to nearly all substantial applications. Thus, the technical challenges of machine learning systems are largely general distributed-application challenges, and these can be largely addressed by managing machine learning systems in a single place as intelligent applications. In addition, choosing the *right* place to manage these applications will make it easier to solve the aforementioned commercial and social challenges as well.

The idea that it is desirable to manage as many components as possible of a data-processing or machine-learning systems under a single control plane is not new: pockets of industry have seen brief, intense, and ultimately unsuccessful attempts to generalize and repurpose specialized schedulers and infrastructure (e.g., trying to run low-latency streaming applications on a batch scheduler built on a distributed storage system). This section will introduce Kubernetes (Burns et al., 2016), a resource manager that is suitable for application development and deployment, the compute and storage workloads that machine learning systems depend on, and thus entire intelligent applications.

Kubernetes runs services and jobs in *Linux containers* (Soltesz et al., 2007), which provide a lightweight isolation mechanism (in order to protect services from other processes that might be running on the same host). Since an important part of isolation is ensuring that unrelated services cannot access the same files, container runtimes also provide support for creating immutable filesystem images that package application code with the libraries it requires. These images are an increasingly important distribution mechanism for open-source and proprietary software alike. More complex distributed applications are typically developed on Kubernetes in a *microservice architecture*, which incorporates several stateless processes, each running in a container and communicating with its peers via message passing.[3]

Microservice architectures have several technical benefits, which we will discuss. However, the *social* benefits of microservice architectures are also interesting in the context of the teams that build machine learning systems. By focusing on microservices, small subteams are freed to focus on narrower spheres of responsibility and merely need to satisfy interface contracts for their services to work with the rest of the system.

The first and most important advantage of Kubernetes for intelligent applications and machine learning systems is the concept of *declarative deployments*: a user publishes an application for Kubernetes not by providing steps in a recipe to execute, but by describing the services that will comprise the system, the resources they will need, and how they are connected to one another. As a consequence, Kubernetes itself is able to ensure that a deployment is as expected and change the state of the system to correct any errors or failures. These deployments also make systems reproducible and portable: the same deployment specification can run

[3]See Dragoni et al. (2017) for a survey of microservice architectures.

on a Kubernetes cluster on a personal workstation, inside a corporate datacenter, or on any public cloud infrastructure.

A crucial difference between Kubernetes and classic HPC schedulers is that Kubernetes provides the primitives to support a productive developer experience. Stateless container architectures enable *continuous integration*, in which an application can be tested in isolation in an exact replica of the production environment after each code change is committed to source control, and *continuous deployment*, in which builds that pass integration testing are automatically pushed out to the production environment and transparently replace older versions. A more sophisticated deployment strategy, called *blue-green deployments* routes some requests or sessions to the new version of the application and the remainder of requests to the old version, shifting the proportion over time to send more requests to the new version if no errors are observed and enabling upgrades without downtime (and speedy rollback if necessary).

These automated testing and deployment advantages apply to conventional applications and also to machine learning systems. The predictive components of intelligent applications are also services that must be built, tested, and deployed, and production model training pipelines can be implemented using the machinery of application build pipelines. Because models can fail silently, we need to be able to route prediction requests to older versions if we observe degraded application performance after installing a new model, just as we need to be able to roll back a misbehaving microservice that passes tests but crashes in production.

Data scientists can benefit from the same user experience that Kubernetes presents to application developers. In particular, data scientists often use *interactive notebook software* (Kluyver et al., 2016), which combines code, documentation, execution, and output in a single document. The promise of notebooks is to support development, communication, and reproducible research, but the latter is only realized with a disciplined data scientist, since a notebook's results may depend on details of the user's environment. By publishing container images with notebooks and dependencies, data scientists can ensure truly reproducible research. By specifying their requirements as part of a declarative deployment, it is possible to establish a predictable research environment on any infrastructure.[4]

It is even possible to use continuous integration and deployment tooling to automatically transform notebooks that define a machine learning pipeline into production services.[5] By supporting tooling to translate directly from a data scientist's preferred environment to a production-ready service, continuous integration tooling can dramatically increase the velocity of development teams; in a traditional model, data scientists use notebooks as a communication tool addressed to developers who reimplement the essential techniques as service endpoints. Manual reimplementation adds time and human effort to the process, but it also means that the quality of model services is dependent on the discipline and understanding of the developer; generating these services automatically ensures that they will always perform error-checking and input sanitization, and that they will always publish model metrics, without specific practitioner effort.

Monitoring and observability are challenges for microservice systems and intelligent applications alike, since one needs to understand the interactions between components to even begin to understand the behavior of a distributed system. The Kubernetes ecosystem supports several log-aggregation services, a time-series database for metrics data, and tooling to visualize and trigger alerts based on log or metric events. By tracking metrics about each stage of our data pipelines and about our predictive models, we can identify concept drift and related problems even in situations where we don't know the ground truth, simply by identifying divergence in the distributions of the metrics we've collected.

Finally, using a single control plane makes machine learning systems easier to manage: instead of tracking metrics and quotas from separate compute, storage, and application clusters, or instead of defining three different scheduling policies in three different languages on three different systems, we can manage every component in Kubernetes. Kubernetes was designed to manage conventional web applications, but it has proven flexible enough to manage scale-out analytic processing workloads like Apache Spark (Zaharia et al., 2016), tightly-coupled HPC-style parallel compute workloads (Sergeev and Del Balso, 2018), and a variety of scalable storage and messaging systems.

[4]See mybinder.org for such a service built on Kubernetes.
[5]See https://github.com/willb/nachlass.

V. Systems and use cases

In this paper, we have argued that many of the challenges of machine learning systems are more tractable when you structure them as intelligent applications on a flexible platform like Kubernetes. Part of the evidence for this argument lives outside this paper: it is real systems and applications that ultimately show that Kubernetes is suitable for machine learning systems. Case studies of machine learning systems on Kubernetes are regularly described at industry conferences. As a recent example, there was a special track devoted to operationalizing machine learning at the Open Source Conference in 2019 (O'Reilly and Associates, 2019); speakers described numerous machine learning systems on Kubernetes, including use cases in finance, developer infrastructure, and broadcast media.

The radanalytics.io community has developed several projects to aid developing and deploying intelligent applications on Kubernetes, including tooling to deploy an application along with a Spark cluster specific to that application as part of a continuous integration and build pipeline.

The Open Data Hub is an intelligent application that provides an on-demand multitenant discovery environment on Kubernetes, including storage, compute, notebook hosting, and monitoring. The Valeria system developed at Université Laval also provides a similar discovery environment on Kubernetes.

The Kubeflow project provides tooling to make machine learning workflows scalable and repeatable on Kubernetes, including notebook environments, experiment management, "auto-ML" functionality (Zhou et al., 2019), and more.

References

L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, Apr. 2016.

M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and ulterior software engineering*. Springer, 2017, pp. 195–216.

J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

A. Nelson, T. Menzies, and G. Gay, "Sharing experiments using open-source software," *Software: Practice and Experience*, vol. 41, no. 3, pp. 283–305, 2011.

O'Reilly and Associates. (2019) OSCON MLOps track.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2503–2511.

A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in Tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," in *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 275–287.

M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.

J. Zhou, A. Velichkevich, K. Prosvirov, A. Garg, Y. Oshima, and D. Dutta, "Katib: A distributed general automl platform on kubernetes," in *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, 2019, pp. 55–57.

William Benton received the Ph.D. degree in computer sciences from the University of Wisconsin in 2008. Since then, he has held a variety of product engineering and emerging-technology roles at Red Hat, Inc. He is currently an engineering manager and a senior principal software engineer in Red Hat's Office of the CTO, focusing on machine learning workloads and workflows in the cloud.