On Challenges in Machine Learning Model Management

Sebastian Schelter, Felix Biessmann, Tim Januschowski,
David Salinas, Stephan Seufert, Gyuri Szarvas

{sseb,biessman,tjnsch,dsalina,seufert,szarvasg}@amazon.com

Amazon Research

Abstract

The training, maintenance, deployment, monitoring, organization and documentation of machine learning (ML) models – in short model management – is a critical task in virtually all production ML use cases. Wrong model management decisions can lead to poor performance of a ML system and can result in high maintenance cost. As both research on infrastructure as well as on algorithms is quickly evolving, there is a lack of understanding of challenges and best practices for ML model management. Therefore, this field is receiving increased attention in recent years, both from the data management as well as from the ML community. In this paper, we discuss a selection of ML use cases, develop an overview over conceptual, engineering, and data-processing related challenges arising in the management of the corresponding ML models, and point out future research directions.

1 Introduction

Software systems that learn from data are being deployed in increasing numbers in industrial application scenarios. Managing these machine learning (ML) systems and the models which they apply imposes additional challenges beyond those of traditional software systems [18, 26, 10]. In contrast to textbook ML scenarios (e.g., building a classifier over a single table input dataset), real-world ML applications are often much more complex, e.g., they require feature generation from multiple input sources, may build ensembles from different models, and frequently target hard-to-optimize business metrics. Many of the resulting challenges caught the interest of the data management research community only recently, e.g., the efficient serving of ML models, the validation of ML models, or machine learning-specific problems in data integration. A major issue is that the behavior of ML systems depends on the data ingested, which can change due to different user behavior, errors in upstream data pipelines, or adversarial attacks to name just some examples [3]. As a consequence, algorithmic and system-specific challenges can often not be disentangled in complex ML applications. Many decisions for designing systems that manage ML models require a deep understanding of ML algorithms and the consequences for the corresponding system. For instance, it can be difficult to appreciate the effort of turning raw data from multiple sources into a numerical format that can be consumed by specific ML models – yet this is one of the most tedious and time consuming tasks in the context of ML systems [32].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In this paper, we introduce a set of intricate use cases in Section 2, and elaborate on the corresponding general challenges with respect to model management. We discuss conceptual challenges in Section 3.1, e.g., how to exactly define the model to manage [35], how to validate the predictions of a model both before and after deployment [3], and how to decide when to retrain a model. Next, we elaborate on data-management related challenges in Section 3.2. These evolve around the fact that ML models are part of larger ML pipelines, which contain the corresponding feature transformations and related metadata [31, 34]. Efficient model management requires us to be able to capture and query the semantics, metadata and lineage of such pipelines [37, 27, 36]. Unfortunately, this turns out to be a tough problem as there is no declarative abstraction for ML pipelines that applies to all ML scenarios. Finally, we list a set of engineering-related challenges, which originate from a lack of agreed-upon best practices as well as from the difficulties of building complex systems based on components in different programming languages (Section 3.3).

2 Selected Use Cases

Time Series Forecasting. Large-scale time series prediction or *forecasting* problems have received attention from the ML, statistics and econometrics community. The problem of forecasting the future values of a time series arises in numerous scientific fields and commercial applications. In retail, an accurate forecast of product demand can result in significant cost reductions through optimal inventory management and allocation. In modern applications, many time series need to be forecasted for simultaneously. Classical forecasting techniques, such as ARIMA models [7], or exponential smoothing and its state-space formulation [14] train a single model per time series. Since these approaches are typically fast to train [33, 15], it is possible to retrain the models every time a new forecast needs to be generated. In theory, this would mean that little to no management of the resulting models is required; however, the practice very often differs significantly. While classical forecasting models excel when time series are well-behaved (e.g., when they are regular, have sufficiently long history, are non-sparse), they struggle with many characteristics commonly encountered in real-world use cases such as cold-starts (new products), intermittent time series and short life cycles [30]. Such complications require models that can transfer information across time series. Even for well-behaved time series, we may often want to be able to learn certain effects (e.g., the promotional uplift in sales) across a number of products. Therefore, forecasting systems in the real-world become quite complex even if the classical models at its core are simple [6]. Maintaining, managing, and improving the required ML pipelines in such systems is a non-trivial challenge – not only in production environments which may require complex ensembles of many simple models, but especially in experimental settings when different individual models are evaluated on different time series. We found that a natural alternative to such complex ensembles of simple models is end-to-end learning via deep learning models for forecasting [11]. Neural forecasting models have the ability to learn complex patterns across time series. They make use of rich metadata without requiring significant manual feature engineering effort [13], and therefore generalize well to different forecasting use cases. Modern, general-purpose implementations of neural forecasting mdoels are available in cloud ML services such as *AWS SageMaker* [17]. However, when such neural forecasting models are deployed in long-running applications, careful management of the resulting models is still a major challenge, e.g., in order to maintain backwards compatibility of the produced models.

Missing Value Imputation. Very often, ML is leveraged to automatically fix data quality problems. A prominent issue in this context are missing values. A scenario in which missing values are problematic are product catalogs of online retailers. Product data needs to be complete and accurate, otherwise products would not be found by customers. Yet in many cases, product attributes may not be entered by sellers, or they might apply a different schema and semantic, which results in conflicting data that would need to be discarded. Manually curating this data does not scale, hence automatic solutions leveraging ML technology are a promising option for ensuring high data quality standards. While there are many approaches dealing with missing values, most

of these methods are designed for matrices only. In many real-world use cases however, data is not available in numeric formats but rather in text form, as categorical or boolean values, or even as images. Hence most datasets must undergo a feature extraction process that renders the data amenable to imputation. Such feature extraction code can be difficult to maintain, and not every data engineer that is facing missing values in a data pipeline will necessarily be able to implement or adapt such code. This is why implementing the feature extraction steps and the imputation algorithms in one single pipeline (that is learned end-to-end) greatly simplifies model management [4] for end users. An end-to-end imputation model that uses hyperparameter optimization to automatically perform model selection and parameter tuning can be applied and maintained by data engineers without in depth understanding of all algorithms used for feature extraction and imputation.

Content Moderation. Another broad set of ML-related tasks can be summarized under the umbrella of content moderation. There is a wide range of such use cases, some simple in their nature (e. g., the detection of swear words in content), while others are more complex, e. g., the detection of fake news, or the detection of copyright infringement in music or video. As an example, we focus onto user comments in online communities [24]. Before publishing user provided content, moderation might take place, often via a mixture of ML methods and human curators. Common tasks are to mark content which contains inappropriate language or does not adhere to a community standard. Such moderation consists of offline training of models using manually labelled data. However, during the production usage of such models, we need to constantly monitor the model accuracy. A typical approach looks as follows: content that can be classified with high accuracy by automated models does not require a human interaction. However, content for which the model output is inconclusive (for this, probabilistic classifiers are of utmost importance) are directly passed to human annotators. This data can then be used to re-train the model online in an active learning setting. Depending on their overall capacity, we can also randomly select sample content that was classified with high probability. Periodically, we should also select and relabel a completely random subset of the incoming content. In this way, we constantly update the model and improve its performance.

Automating Model Metadata Tracking. An orthogonal data management use case of great importance in all ML applications is tracking the metadata and lineage of ML models and pipelines. When developing and productionizing ML models, a major proportion of the time is spent on conducting model selection experiments, which consist of training and tuning models and their corresponding features [29, 6, 18, 26, 40, 3]. Typically, data scientists conduct this experimentation in their own ad-hoc style without a standardized way of storing and managing the resulting experimentation data and artifacts. As a consequence, the models resulting from these experiments are often not comparable, since there is no standard way to determine whether two models had been trained on the same input data. It is often technically difficult to reproduce successful experiments later in time. To address the aforementioned issues and assist data scientists in their daily tasks, we built a lightweight system for handling the metadata of ML experiments [27]. This system allows for managing the metadata (e. g., *Who created the model at what time? Which hyperparameters were used? What feature transformations have been applied?*) and lineage (e. g., *Which dataset was the model derived from? Which dataset was used for computing the evaluation data?*) of produced artifacts, and provides an entry point for querying the persisted metadata. Such a system enables regular automated comparisons of models in development to older models, and thereby provides a starting point for quantifying the accuracy improvements that teams achieve over time towards a specific ML goal. The resulting database of experiments and outcomes can furthermore be leveraged later for accelerating meta learning tasks [12].

3 Model Management Challenges

The aforementioned use cases lead to a number of challenges that we discuss in the following subsections. We categorize the challenges broadly into three categories: (i) *conceptual challenges*, involving questions such as what is part of a model, (ii) *data management challenges*, which relate to questions about the abstractions used in ML pipelines and (iii) *engineering challenges*, such as building systems using different languages and specialized hardware.

3.1 Conceptual Challenges

Machine Learning Model Definition. It is surprisingly difficult to define the actual model to manage. In the most narrow sense, we could consider the model parameters obtained after training (e.g., the weights of a logistic regression model) to be the model to manage. However, input data needs to be transformed into the features expected by the model. These corresponding feature transformations are an important part of the model that needs to be managed as well. Therefore, many libraries manage ML *pipelines* [25, 22] which combine feature transformations and the actual ML model in a single abstraction. There are already established systems for tracking and storing models which allow to associate feature transformations with model parameters [35, 37, 27]. Due to the i.i.d.-assumption inherent in many ML algorithms, models additionally contain implicit assumptions on the distribution of the data on which they are applied later. Violations of these assumptions (e.g., covariate shift in the data) can lead to decreases in prediction quality. Capturing and managing these implicit assumptions is tricky however, and systematic approaches are in the focus of recent research [3, 16, 28]. A further distinction from a systems perspective is whether the model is considered to be a 'black-box' with defined inputs and outputs (e.g., a docker image with a REST-API or a scikit-learn pipeline with transformers and estimators) or whether the model is presented in declarative form comprised of operations with known semantics (e.g., the computational graph of a neural network). Even along these outlined dimensions, the heterogeneity of ML models and the complexity of many real-world ML applications render the definition of ML models difficult. For example: How should one handle combinations of models (e.g., in ensembles when models from different languages or libraries are combined, or when an imputation model is part of the preprocessing logic of another model) or meta-models, e.g., meta-learning algorithms or neural architecture searches.

Model Validation. The ability to back-test the accuracy of models over time is crucial in many real-world usages of ML. Models evolve continuously as data changes, methods improve or software dependencies change. Every time such a change occurs, model performance must be re-validated. These validations introduce a number of challenges and pitfalls: When we compare the performance of ML models, we must ensure that the models have been trained and evaluated using the same training, test and validation set. Additionally, it is crucial that the same code for evaluating metrics is used throughout time and across different models to be able to guarantee comparability. The more experiments we conduct however, the more we risk to overfit on the test or validation data. A further practical problem in backtesting comes from long-running experiments and complex interactions between pipelines of models. A change in backtesting results is often hard to attribute to specific code and configuration changes. Furthermore, in long-existing, highly-tuned ML systems, it is unlikely that the overall accuracy improves significantly. Usually, a new model introduces an improvement but this improvement comes at some cost (such as longer prediction times), which makes wrapping-up release candidates challenging. Specific domains introduce additional challenges. For instance, the train/validation/test split that is required in virtually all ML evaluations is often conducted by randomly partitioning an initial dataset into given proportions, such as 80%, 10% and 10% of the data. In real-world scenarios, in particular in those where the i.i.d.-assumption often made in ML does not hold, we must partition the data more carefully to avoid leakage. For instance in the case of forecasting, it would not be appropriate to split the data per time-series. We need to make sure that no future information such as seasonality is revealed on a training set. Further practical considerations are

to consider dynamic backtesting scenarios where we conduct rolling evaluations over time to ensure that our backtest results do not depend on a specific point in time where we performed the split. Each evaluation should track such information in order to always be able to assert that no data leakage happened in a previous evaluation.

In other use cases that involve classification models, such as the imputation of missing values, naive train/test splits can lead to problems as well. Real world datasets often have very skewed distributions of categories. Few categories are observed very often, but many categories are observed only very rarely. Conducting a naive cross-validation split often results in training sets or test sets that contain not enough data for single categories to either train a sensible model or to get a reasonable estimate of the generalization error.

Apart from validation of models in offline backtests, assessing the performance of models in production is crucial. Making implicit assumptions about data explicit and enabling automatic data validation [28] is even more important in online scoring scenarios. When launching a new model, we can validate its effectiveness in various degrees. Launching the model in "shadow mode" (i.e., routing part of the data to the model, but not consuming its output) often helps to catch operational problems. Whenever possible, one should conduct rolling comparisons between ground truth and predictions. A/B or bandit tests are often employed to explose a model in an end-user experience. Note that for forecasting, A/B tests can be quite challenging due to the long and indirect feedback cycles. Independent of the model launch mechanism, sanity-checking the model output is best practice. Since having high-quality sanity checks in place is as difficult as solving the original ML problem typically, we can employ simple, cheap-to-run or robust prediction models for comparisons, work with thresholding or randomized human inspection. If the core models allows to quantify its reliability via probabilities, this helps in making more informed choices on the model validity.

Decisions on Model Retraining. Deciding when to retrain a model is challenging. Training usually is done offline and models are loaded at prediction/inference time in a serving system. If new events occur that our models have not been trained on, we need to trigger retraining or even re-modelling. In the retail demand forecasting domain, examples of such events are the introduction of a new public holiday, a new promotional activity or a weather irregularity. Detecting change in the data is critical, as issues and changes in datasets drastically affect the quality of the model and can invalidate evaluations. When such data changes impact the final metrics (as noticed through backtests), backtracking the root-cause to the dataset is only possible when dataset metadata and lineage have been recorded beforehand. Similar challenges occur in the imputation use case. Standard cross-validation techniques help to estimate the generalization error and tune precision thresholds for missing value imputation models. But in the presence of noisy training data, it is important to always have some fraction of the predictions audited by human experts and mark imputations as such. When humans are in the loop of such a system, interesting model management challenges can emerge. If auditors are able to update the training data with their audited data, they may often tend to solidarize with the ML pipeline in that they try to increase the model performance as much as they can, which can lead to severe overfitting. As a consequence, everyone involved in the model lifecycle should have a minimal understanding of the caveats of ML systems, such as overfitting. Furthermore, such cases show that it is important to carefully control evaluation and training samples, and guaranteeing this in continuously changing data pipelines can be quite challenging.

Adversarial Settings. In our exemplary scenario for content moderation in communities, an adversarial setting can occur where malicious users try to understand the boundary conditions of the classifier. For example, users could try to reverse engineer the thresholds under which content is automatically classified and then try to exploit weaknesses of the classifier. In such settings, incoming data will shift over time, which requires careful monitoring of the input and the distribution of the output probabilities. We would assume that more and more content contributions are close to the prediction threshold so that the distribution shifts over time. Once such a distribution shift is deemed to be severe enough, one needs to retrain the model and for this, must obtain new labeled data. Another approach is to rely on active learning. Given a model that outputs a probabilistic classification, it can be constantly updated by acquiring (manually labeled) data from both the decision boundary as

well as randomly picked instances. Modeling the adversarial actions themselves by another model is possible in theory and certainly desirable as it eliminates the need to keep labelling data, but in practice, this is often extremely difficult, therefore careful monitoring of the deployed models is crucial.

3.2 Data Management Challenges

Lack of a Declarative Abstraction for the whole ML Pipeline. A major success factor of relational databases is that that their data processing operations are expressed in a declarative language (SQL) with algebraic properties (relational algebra). This allows the system to inspect computations (e.g., for recording data provenance) and to rewrite them (e.g., for query optimization). Unfortunately, there is a lack of a declarative abstraction for end-to-end ML pipelines. These comprise of heterogenous parts (e.g., data integration, feature transformation, model training) which typically apply different operations based on different abstractions. Loading and joining the initial data applies operations from relational algebra. Feature transformations such as one-hot-encoding categorial variables, feature hashing, and normalization often apply map-reduce like transformations, composed in a 'pipeline'-like abstraction popularized by scikit-learn [25, 22, 31]. The actual model training phase uses mostly operations from linear algebra which in many cases require specialized systems and/or hardware to be run efficiently [2, 9]. While general dataflow systems such as Apache Spark support all of the listed phases, it turns out that the dataflow abstraction is very inefficient for many advanced analytics problems [21, 5] and most models are implemented as black-boxes in such systems. This led to the development of specialized systems, such as engines for training deep neural networks, that are tailored towards certain learning algorithms (mini-batch stochastic gradient descent) and very efficiently leverage specialized hardware such as GPUs. Unfortunately, these systems do not support general relational operations, e.g., they lack join primitives. As a consequence, different systems need to be "glued" together in many real-world ML deployments, and often require external orchestration frameworks that coordinate the workloads. This situation has tremendous negative effects for model management tasks: It complicates the extraction of metadata corresponding to a particular pipeline (e.g., because the hyperparameters of feature transformations are contained the relational part of pipeline, while hyperparameters for the model training part are found in the computational graph of a neural network), it makes replicability and automation of model selection difficult as many different systems have to be orchestrated, and as a consequence makes it hard to automate model validation.

Querying Model Metadata. In order to automate and accelerate model lifecycle management, it is of crucial importance to understand metadata and lineage of models (e.g., their hyperparameters, evaluation scores as well as the datasets on which they were trained and validated). This metadata is required for comparing models in order to decide which one to put into production, for getting an overview of team progress, and for identifying pain points as a basis for deciding where to invest tuning efforts. Additionally, a centralized metadata store forms a basis for accelerating learning processes (e.g., through warm-starting of hyperparameter search [12]) and is also helpful for automating certain simple error checks (e.g., "did someone accidentally evaluate on their training set?"). Unfortunately, many existing ML frameworks have not been designed to automatically expose their metadata. Therefore, metadata capturing systems [35, 37, 27] in general require that data scientists instrument their code for capturing model metadata, e.g., to annotate hyperparameters of feature transformations or data access operations. Unfortunately, we found that data scientists often forget to add these annotations or are reluctant to spend the extra effort this imposes. In order to ease the adoption of metadata tracking systems, we explore techniques to automatically extract model metadata from ML applications. In our experience, this works well, as long as certain common abstractions are used in ML pipelines, e.g., 'data frames' in pandas or Spark which hold denormalized relational data, and pipelines (in scikit-learn and SparkML) which comprise a way to declare complex feature transformation chains composed of individual operators. For applications built on top of these abstractions, it is relatively simple to automate metadata tracking via reflection and inspection

of these components at runtime [27]. In applications not built from declarative components (e.g., systems that rely on shell-scripts to orchestrate model training) on the other hand, it is very hard to automate the metadata extraction process, and a custom component that exposes the metadata has to be built manually. An extension of this problem is that for debugging purposes, often intermediate model outputs need to be inspected [39, 36], in addition to model metadata and lineage.

3.3 Engineering Challenges

Multi-Language Code Bases. A hard-to-overcome practical problem in model management originates from the fact that end-to-end ML applications often comprise of components written in different programming languages. Many popular libraries for ML and scientific computations are written in python, often with native extensions. Examples include scikit-learn [25], numpy and scipy as well as the language frontends of major deep learning systems such as MXNet [9], Tensorflow [2] or PyTorch [23]. For data pre-processing, JVM-based systems such as Apache Spark [38] are often preferred, due to static typing in the language and better support for parallel execution. Such heterogeneous code bases are often hard to keep consistent as automatic refactoring and error checking tools can only inspect either the python or the JVM part of the code, but will not be able to tackle problems across the language barrier. This problem has been identified as "Multiple-Language Smell" [29] in the past already. Furthermore, such code bases are hard to deploy later on, as they require setups with many different components that need to be orchestrated. E.g., a Spark cluster must be spun up for pre-processing, afterwards the data must be moved to a single machine for model training, and the cluster must be torn down afterwards. An orthogonal problem is the efficient and reliable exchange of data between the components of the system written in different languages. Often, this happens via serialization to a common format on disk, which must be kept in sync. In the future, language-independent columnar memory formats such as Apache Arrow for data or ONNX for neural networks [1] are promising developments to address this issue.

Heterogeneous Skill Level of Users. Teams that provide ML technology often have to support users that come from heterogeneous backgrounds. Some user teams have a strong engineering culture, are maintaining large-scale data pipelines and have many job roles dedicated to highly specific tasks. On the other hand, there are product teams with fewer or no members in technical roles. And there are researchers with little experience in handling production systems. ML products should be usable for all customers along this spectrum of technical skills and ensure a smooth transition from one use case to another. The broad range of use cases also requires one to build ML models that work well for large and small datasets at the same time - although these often require different approaches. Some larger datasets can only be tackled with specialized hardware, such as GPUs. But the methods that excel at these tasks with large datasets are often suboptimal when dealing with small data. E.g, if initial training datasets are manually curated by human experts, there is no point in using complex neural network models. In our experience, robust but simple linear models with carefully designed features often work better in these cases. Another dimension that distinguishes the various use cases is how the ML models are deployed afterwards. Some teams might run training and serving using scheduled jobs, but smaller teams often do not want to maintain this type of infrastructure. As a consequence, models have to be designed such that they can be deployed in large-scale scheduled workflows but also quickly tested in experimental prototypes.

Backwards Compatibility of Trained Models. Systems such as Sagemaker [17] or OpenML [35] require backwards compatibility of ML models. A model that was trained last month or last year should still be working today, in particular when trained models might be used in production deployments. Various degrees of backwards compatibility can be satisfied. We may want to impose that the *exact* same result can be obtained, that a *similar* result can be achieved, or in the weakest case that the model can still run. Ensuring those conditions reveals several challenges on model management. Clearly, when a model is deployed and serves predictions, making sure that the exact same result can be retrieved is a strict requirement. To this end, storing all the

components defining a model is fundamental in being able to guarantee backwards compatibility and thereby long-term usage. One needs to store at least the data transformations that were required, the implementation of the algorithm, its configuration and finally all its dependencies which can be done by storing model container versions. In the case of training, ensuring that the exact result can be obtained over time may be too strict. This would prevent opportunities to improve models and may be hard to achieve in systems where execution is not fully deterministic (e.g., distributed systems or neural network models when they rely on GPU computation). In such cases, one may prefer to ensure that training results are similar (or better) over time.

4 Conclusion

We introduced a set of ML use cases and discussed the resulting general challenges with respect to model management. Our focus was on conceptual challenges, challenges related to data-management and engineering. Our experience confirms the need for research on data management challenges for model management and ML systems in general [29, 18]. The problems in this space have their roots in the inherent complexity of ML applications, the lack of a declarative abstraction for end-to-end ML, and the heterogeneity of the resulting code bases. The recent success of deep neural networks in the ML space is often attributed to the gains in prediction accuracy provided by this class of models. We think however that another success factor of neural networks is often overlooked. Neural networks provide an algebraic, declarative abstraction for ML: they are built from computational graphs comprised of tensor operators with clearly defined semantics. Similar to query plans built from relational operators in SQL, this allows for easy combination of complex models and for automatic optimization of the execution on different hardware. With the rise of symbolic neural network APIs it became much simpler for practitioners to experiment with new models, which was one of the main factors contributing to the recent advances we have seen in this field. We strongly feel that the ML space can greatly benefit from adopting proven principles from the data management community such as declarativity and data independence.

Additionally, we observe the rediscovery of many best practices from software engineering and their adaption to managing ML models [40]. Examples are scores for quantifying the production-readiness of an ML system [8] or unit-tests for the implicit data assumptions inherent in ML models [28]. A promising research direction here is to tackle the challenges stemming from the lack of a common declarative abstraction for end-to-end ML, and to design new languages that support relational algebraic and linear primitives at the same time (and can reason about them them holistically [19, 20]). Finally, we would like to point out that questions of model management encounter a growing interest in the academic community. Examples of venues for related research are the 'Systems for Machine Learning' workshop[1] at the NIPS, ICML and SOSP conferences, the 'Data Management for End-to-End Machine Learning' workshop[2] at ACM SIGMOD, and the newly formed SysML conference[3].

References

[1] ONNX open neural network exchange format, 2017.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. *OSDI*, 16:265–283, 2016.

[3] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *KDD*, pages 1387–1395, 2017.

[1] `http://learningsys.org`
[2] `http://deem-workshop.org`
[3] `http://sysml.cc`

[4] Felix Biessmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. "deep" learning for missing value imputationin tables with non-numerical data. *CIKM*, 2018.

[5] Christoph Boden, Tilmann Rabl, and Volker Markl. Distributed machine learning-but at what cost? *NIPS Workshop on Machine Learning Systems*, 2017.

[6] Joos-Hendrik Böse, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Dustin Lange, David Salinas, Sebastian Schelter, Matthias Seeger, and Yuyang Wang. Probabilistic Demand Forecasting at Scale. *PVLDB*, 10(12):1694–1705, 2017.

[7] George E P Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[8] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. Whats your ml test score? a rubric for ml production systems. *NIPS Workshop on Reliable Machine Learning in the Wild*, 2016.

[9] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[10] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. *NSDI*, pages 613–627, 2017.

[11] Christos Faloutsos, Jan Gasthaus, Tim Januschowski, and Yuyang Wang. Forecasting big time series: old and new. *PVLDB*, 11(12):2102–2105, 2018.

[12] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. *AAAI*, pages 1128–1135, 2015.

[13] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *arXiv preprint arXiv:1704.04110*, 2017.

[14] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

[15] Rob J Hyndman, Yeasmin Khandakar, et al. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University, Department of Econometrics and Business Statistics, 2007.

[16] Nick Hynes, D Sculley, and Michael Terry. The data linter: Lightweight, automated sanity checking for ml data sets. *NIPS Workshop on Machine Learning Systems*, 2017.

[17] Tim Januschowski, David Arpin, David Salinas, Valentin Flunkert, Jan Gasthaus, Lorenzo Stella, and Paul Vazquez. Now available in amazon sagemaker: Deepar algorithm for more accurate time series forecasting. *https://aws.amazon.com/blogs/machine-learning/now-available-in-amazon-sagemaker-deepar-algorithm-for-more-accurate-time-series-forecasting/*, 2018.

[18] Arun Kumar, Robert McCann, Jeffrey Naughton, and Jignesh M Patel. Model Selection Management Systems: The Next Frontier of Advanced Analytics. *SIGMOD Record*, 2015.

[19] Andreas Kunft, Alexander Alexandrov, Asterios Katsifodimos, and Volker Markl. Bridging the gap: towards optimization across linear and relational algebra. *SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, 2016.

[20] Andreas Kunft, Asterios Katsifodimos, Sebastian Schelter, and Volker Markl. Blockjoin: efficient matrix partitioning through joins. *PVLDB*, 10(13):2061–2072, 2017.

[21] Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! but at what cost? *HotOS*, 15:14–14, 2015.

[22] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *JMLR*, 17(34):1–7, 2016.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[24] John Pavlopoulos, Prodromos Malakasiotis, and Ion Androutsopoulos. Deeper attention to abusive user content moderation. *EMNLP*, pages 1125–1135, 2017.

[25] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[26] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. Data management challenges in production machine learning. *SIGMOD*, pages 1723–1726, 2017.

[27] Sebastian Schelter, Joos-Hendrik Boese, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. Automatically Tracking Metadata and Provenance of Machine Learning Experiments. *NIPS Workshop on Machine Learning Systems*, 2017.

[28] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating large-scale data quality verification. *PVLDB*, 11(12):1781–1794, 2018.

[29] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *NIPS*, pages 2503–2511, 2015.

[30] Matthias Seeger, David Salinas, and Valentin Flunkert. Bayesian intermittent demand forecasting for large inventories. *NIPS*, pages 4646–4654, 2016.

[31] Evan R Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J Franklin, and Benjamin Recht. Keystoneml: Optimizing pipelines for large-scale advanced analytics. *ICDE*, pages 535–546, 2017.

[32] Michael Stonebraker and Ihab F Ilyas. Data integration: The current status and the way forward. *Data Engineering Bulletin*, 41(2):3–9, 2018.

[33] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 2017.

[34] Tom van der Weide, Dimitris Papadopoulos, Oleg Smirnov, Michal Zielinski, and Tim van Kasteren. Versioning for end-to-end machine learning pipelines. *SIGMOD Workshop on Data Management for End-to-End Machine Learning*, page 2, 2017.

[35] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[36] Manasi Vartak, Joana M F da Trindade, Samuel Madden, and Matei Zaharia. Mistique: A system to store and query model intermediates for model diagnosis. *SIGMOD*, pages 1285–1300, 2018.

[37] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. Model db: a system for machine learning model management. *SIGMOD Workshop on Human-In-the-Loop Data Analytics*, page 14, 2016.

[38] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI*, 2012.

[39] Zhao Zhang, Evan R. Sparks, and Michael J. Franklin. Diagnosing machine learning pipelines with fine-grained lineage. *HPDC*, pages 143–153, 2017.

[40] Martin Zinkevich. Rules of machine learning: Best practices for ml engineering, 2017.