Data Management in Machine Learning: Challenges, Techniques, and Systems

Arun Kumar
UC San Diego
La Jolla, CA, USA

Matthias Boehm
IBM Research – Almaden
San Jose, CA, USA

Jun Yang
Duke University
Durham, NC, USA

ABSTRACT

Large-scale data analytics using statistical machine learning (ML), popularly called advanced analytics, underpins many modern data-driven applications. The data management community has been working for over a decade on tackling data management-related challenges that arise in ML workloads, and has built several systems for advanced analytics. This tutorial provides a comprehensive review of such systems and analyzes key data management challenges and techniques. We focus on three complementary lines of work: (1) integrating ML algorithms and languages with existing data systems such as RDBMSs, (2) adapting data management-inspired techniques such as query optimization, partitioning, and compression to new systems that target ML workloads, and (3) combining data management and ML ideas to build systems that improve ML lifecycle-related tasks. Finally, we identify key open data management challenges for future research in this important area.

1. INTRODUCTION

Analysis of large datasets using statistical machine learning (ML) algorithms, popularly called advanced or deep analytics, is central to modern data-driven applications in business intelligence (BI), e-commerce, healthcare, science, and other domains [24, 47]. Dating back to the in-RDBMS data mining boom of the late 1990s, the database industry and academia have been working for over a decade on data management-oriented challenges in ML. This has led to a proliferation of systems and frameworks for scalable and fast ML built by our community [2, 3, 38, 46, 70, 109], as well as projects that apply database-inspired ideas to make ML faster and more user-friendly [9, 17, 29, 36, 105, 107]. The diversity of this landscape of systems and projects could be overwhelming for data management researchers, data scientists, and system developers alike.

Goals: This tutorial aims to provide a timely and comprehensive review of systems and techniques that tackle data management challenges in the context of ML workloads. Our focus is on analyzing the technical challenges and on explaining the key ideas, architecture, strengths, and limitations of major systems that address these challenges. This tutorial aims to provide data management researchers and systems developers with a survey of effective techniques and open issues, and to help identify systems they could build upon or compare with. It could also help data scientists understand the assumptions, pros, and cons of different systems and make more informed choices for their applications.

Tutorial Scope: Unlike previous tutorials on ML for "Big Data" [26], we do not focus on a general introduction to ML, usage of ML systems, or general dataflow or graph analytics systems, which have been covered before [10, 103]. Instead, our focus is on identifying general data management challenges and techniques in ML across a broader swathe of works. Given this focus, we also do not cover deep learning algorithms [66] and systems [4, 14, 25, 52]. Deep learning is popular specifically for image, speech, and text data, but is too broad to cover along with other techniques [101], and unlike many other ML workloads, deep learning is typically very compute-intensive and often requires GPUs, FPGAs, or even custom ASICs [4]. We also exclude works that apply ML for text analytics, domain-specific applications, or for improving RDBMS internals such as speculative execution [79], learning cost models [89], predicting workload patterns [80], or resource allocation [68].

Tutorial Outline: This 1.5-hour tutorial covers the following technical content:

- **Workload Characterization:** We motivate a *data-centric view* of ML by providing some basic background on the common data characteristics, data management operations, data access patterns of popular ML algorithms, and pre-processing techniques.
- **ML in Data Systems:** We review systems and frameworks that integrate ML algorithms, frameworks, and languages with existing data systems (Section 2). We will discuss techniques ranging from UDF-centric approaches to deeply integrated approaches.
- **DB-Inspired ML Systems:** We review systems and frameworks designed for ML workloads that apply and adapt DB-inspired techniques (Section 3). Example techniques include optimization with rewrites and operator selection, incremental model maintenance, compression, and access methods.
- **ML Lifecycle Systems:** We review systems that combine DB and ML ideas and target ML lifecycle tasks that go beyond improving the performance of individual algorithms (Section 4).

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA
 ISBN 978-1-4503-4197-4/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3035918.3054775

- **Open Problems:** Finally, we identify and discuss several open research problems as potential directions for new research in this emerging area (Section 5).

Target Audience: This tutorial targets all researchers and practitioners interested in data management challenges and techniques in the context of building scalable and user-friendly systems for ML. We assume that the audience is generally familiar with ML applications and common technical terms. We do not require prior knowledge of specific ML algorithms, workload characteristics, or system internals.

2. ML IN DATA SYSTEMS

We cover systems that integrate ML algorithms with an RDBMS or more recent dataflow systems to bring ML computations closer to where the data resides (e.g., in an RDBMS, or on HDFS). Thus, they avoid or reduce the cost of moving data to specialized ML toolkits. In the tutorial, we will introduce these systems, explain the architecture of representative examples, and highlight the data management challenges that arise, along with how they are tackled.

2.1 UDF-Oriented ML

An early approach to scalable ML exploited the user-defined function (UDF) and user-defined aggregate (UDA) abstractions in data systems. Examples include ATLAS [100], in-RDBMS ML libraries such as Oracle Data Mining, GLADE, which introduced generalized linear aggregates using UDAs [22], one-pass algorithms computing sufficient statistics using UDFs [76], as well as Mahout on Hadoop [1] and MLlib on Spark [70]. Some systems offer more templated approaches to reduce software development effort. Examples include Bismarck [38], which offers a unified architecture based on stochastic gradient descent, and MADlib [24, 46], which provides abstractions for in-RDBMS ML with type bridging from database types such as the SQL:99 `array` type. Similarly, UDFs are also commonly used for parallel prediction [81]. Overall, such systems make it easier to use ML in conjunction with regular SQL for data processing.

2.2 Learning over Joins

Some recent efforts integrate ML with data systems more deeply by optimizing ML over datasets that are logically the output relational queries, especially, joins. This includes Orion [63], which introduced the "factorized learning" technique to push generalized linear models through joins to avoid redundancy in ML computations, Santoku [62], a library of factorized learning and scoring algorithms in R, F [87], a new algorithm and tool for linear regression over "factorized joins," and Morpheus, which generalizes this idea to any ML computations expressible in the formal language of linear algebra [21]. Closely related are LibFM [84], which lets users specify repeating patterns in the data (possibly caused by joins), and TensorDB [58], which pushes tensor decompositions through joins and unions.

2.3 SRL Systems

Some systems exploit RDBMSs to support complex multi-relational ML models known as statistical relational learning (SRL) models [42]. The primary example is DeepDive [31], which supports Markov Logic Networks (MLN) and derived SRL models on top of an RDBMS. DeepDive exploits the advanced join processing capabilities of RDBMSs to scale MLN inference, making it possible to apply such models to large-scale datasets [75]. Another example, ERACER [69], implemented relational dependency network models on top of an RDBMS for data cleaning tasks.

2.4 Query Generators

A further class of systems provide higher level abstractions on top of a data system to simplify the development and customization of new and existing ML algorithms. Under the covers, these systems generate either SQL queries (potentially augmented with procedural extensions and UDFs) on top of an RDBMS, or jobs for data-parallel frameworks such as Hadoop or Spark. The primary examples in this class are the R-based analytics systems such as RIOT-DB [109], Oracle R Enterprise [3], IBM BigR [104], and SparkR [99], which provide R as a front-end, store matrices or data frames in the underlying system, and convert operations over these matrices into queries. Further examples include ScalOps [18], which generates Datalog programs and finally data-parallel jobs, as well as SimSQL [19, 40], which targets Bayesian ML model specifications that generate SQL queries.

2.5 Deep RDBMS Integration

A few projects extend core RDBMS technology to better support ML workloads, in contrast to the previously described classes of systems that leave the data system unaltered. There are two main categories. First, there are DBMS extensions with built-in support for specific model classes. For example, the Fa [34] and F^2DB [39] systems allow declarative forecasting queries by automating model creation, maintenance, and usage. Second, systems like SAP HANA aim to integrate linear algebra for a wide range of ML algorithms. Existing prototypes integrate sparse matrices into SAP HANA's delta architecture [54] and extend the database task scheduler to support multi-threaded OpenMP applications such as linear algebra kernels [102].

3. DB-INSPIRED ML SYSTEMS

We cover a variety of systems and domain-specific languages (DSLs) that go beyond just reusing existing data systems for ML workloads. Many techniques used here are inspired by databases, programming languages, and high performance computing. We will cover a variety of techniques, ranging from optimization and processing, over storage and access methods, to deployment in the cloud. The major differences with traditional RDBMSs are the focus on linear algebra and other ML operations, general DAG structures, and specific sparse and dense data representations.

3.1 Rewrites and Operator Selection

Similar to traditional query optimization, many state-of-the-art optimizing compilers for ML algorithms like RIOT [109], OptiML [94], SystemML [16, 17], Cumulon [48], and Mahout Samsara [86] rely on simplification rewrites [16, 94, 109] and operator selection [17, 48, 86]. Most of these systems use pattern-matching rewrites and a variety of physical operators, chosen with a cost model that incorporates data and cluster characteristics. One important technique—similar to join ordering—is matrix multiplication chain optimization, for which SpMacho incorporated sparsity estimates into a common dynamic programming algorithm [55] and FAQ described a generalization to so-called functional aggregate queries [57]. Furthermore, SystemML SPOOF

made a case for automatic rewrite identification via sum-product optimization on restricted relational algebra plans [35]. Finally, note that initially unknown or changing characteristics require runtime plan adaptation similar to adaptive query processing. SystemML, for instance, adapts plans during runtime via dynamic recompilation [16].

3.2 Incremental Model Maintenance

Incremental maintenance of materialized views is a well studied topic in the database literature. In the spirit of MauveDB's model-based views [32], we can regard an ML model and its predictions as a materialized view of the input data. Several learning tasks then allow efficient incremental maintenance. For example, LINVIEW derives delta update rules from linear algebra programs [73], and incremental iterations aim to reduce the working set across iterations of fixpoint computations [37]. Similarly, DeepDive performs incremental grounding and inference for SRL models [88], Hazy incrementally maintains classification views [59], and Velox applies offline and online learning [27].

3.3 Operator Fusion and Code Generation

Modern in-memory database systems often apply query compilation. Several ML systems also use fused operators or automatic code generation to reduce the number of intermediates and input scans, or to exploit sparsity across operations. SystemML [11, 17] and Cumulon [48] use hand-coded fused operators, whereas [111] applies static analysis and code generation techniques to optimize I/O for matrix computation. OptiML [94] and Emma [7] apply automatic fusion in the context of DSLs embedded in functional programming languages. Tupleware [29] and Kasen [108] generate distributed programs for UDF-centric programs but without exploiting sparsity across operations. SystemML SPOOF [35] recently introduced a holistic framework for automatic rewrite identification and operator fusion, including the generation of sparsity-exploiting operators.

3.4 Asynchronous Execution

A few recent systems go beyond the bulk synchronous processing model to enable asynchronous execution of iterative ML algorithms, especially those that use stochastic gradient descent (SGD). Since SGD is often robust to the order of updates, we can use asynchronous execution to avoid global barriers. Example systems are GraphLab [67] and TensorFlow [4]. In contrast, Hogwild! [74] uses lock-free updates of a shared model for multi-threaded, single-node SGD. They showed, for sparse model updates, that SGD still achieves near-optimal convergence rates. Finally, Gonzalez et al. also extended the Hogwild! idea to dataflow systems [43].

3.5 Compression and Scan Sharing

Many ML algorithms are iterative and perform repeated matrix-vector multiplications. Since matrix-vector multiplications are—similar to traditional table scans—even in-memory I/O-bound, existing work tries to reduce the data size via compression or reduce the number of scans. First, SciDB uses general-purpose compression techniques in the storage manager and decompresses arrays block-wise for each operation [93]. In contrast, SystemML employs lightweight database compression techniques and executes linear algebra operations directly on the compressed matrix representation [36]. Second, once again due to the prominence of I/O-bound operations, other scenarios such as cross validation, ensemble learning, feature selection, and hyper-parameter tuning also typically benefit from scan sharing. For example, MLbase's TUPAQ [91] and Columbus [105] use batching to evaluate multiple model configurations in one pass. Similarly, SystemML uses a technique called runtime piggybacking to batch MR jobs submitted from concurrent threads of a task-parallel `parfor` loop [15] for scan sharing.

3.6 Index Structures and Partitioning

Inspired by traditional access methods in database systems, some ML systems employ dedicated index structures, operation- and topology-aware partitioning and replication, as well as buffer management. First, the LAB-tree (Linearized Array B-tree) [110] indexes out-of-core matrices in a sparsity-aware manner. Similarly, the AT MATRIX (Adaptive Tile Matrix) [56] also uses sparse and dense leaf blocks but is designed for in-memory, NUMA-aware parallelization. The TileDB storage manager uses fragments to handle dense and sparse arrays with random writes [78]. Furthermore, there are domain-specific data structures such as a skip list for I/O-efficient processing of forecasting queries [41]. Second, various systems apply partitioning and replication in an operation- or topology-aware manner. For example, the AT MATRIX [56] uses horizontal range partitioning and socket-local task queues with task-stealing across queues. DimmWitted [107] explores the tradeoff of statistical and hardware efficiency in NUMA architectures via different access methods and replication strategies. ArrayStore [90] investigates effective partitioning schemes for typical array operations and accessing adjacent tiles. Complementary to these strategies, SystemML injects hash partitioning directives for out-of-core matrices that are used read-only in loops to avoid shuffling per iteration [17]. Third, there is also work on buffer management. Elementary [106] explores materialization and buffer management for out-of-core factor graphs, whereas SystemML employs a buffer pool [17] to evict intermediate results of linear algebra programs, if required.

3.7 Cloud ML Resource Elasticity

In public or private clouds, BI and ML workloads share similar challenges of cost-effective resource allocation and robustness against preemption. Cumulon [48] simplifies the deployment in cloud environments by optimizing physical plans for monetary costs under time constraints. This includes allocation decisions such as the cluster size, node types, and configurations. In contrast, SystemML's resource optimizer [50] optimizes for performance without unnecessary over-provisioning via an online what-if analysis, in a plan-aware manner. Similar to SystemML's resource adaptation, Dolphin [23] performs runtime configuration optimization but for parameter servers. Cümülön [49]—an extension of Cumulon that makes use of cheap, but unreliable "spot instances"—considers the risk of preemption to optimize bidding strategies. Narayanamurthy et al. handle preemption in an algorithm-specific manner by approximating the loss function for missing partitions [72]. Similarly, Schelter et al. introduced an optimistic recovery mechanism for fixpoint computations based on compensation functions [85].

4. ML LIFECYCLE SYSTEMS

We cover systems that target—beyond training individual models—other important tasks in the ML lifecycle, in-

cluding tasks that occur before training or as "outer loops" surrounding training and prediction. We discuss tasks such as feature engineering, model selection, and model management. Many of these systems apply DB-centric ideas such as declarativity, interactivity, and optimization, often combining such ideas with techniques from the ML literature.

4.1 Feature Engineering

Feature engineering is the process of constructing features, which includes tasks such as feature extraction from raw data and feature selection [44]. It is often considered to be the most time-consuming part of an applied ML project [33]. While the ML community has studied algorithmic feature selection, auxiliary pains in feature engineering have largely been ignored. Our community is building systems to support such tasks by taking a dataflow-oriented view. Brainwash [9] and DeepDive [83] abstracted such tasks using workflows of UDFs and proposed DB-style ideas to optimize them. Columbus proposed a DSL for exploratory feature selection and applied both DB-style and ML-style optimizations such as batching, QR decomposition, and warm starting to reduce runtimes under accuracy constraints. Zombie [8] optimized feature extraction and refinement tasks by combining indexing and partitioning ideas with a multi-armed bandit technique to read only parts of the data. KeystoneML [92] provides libraries for certain forms of featurization and optimizes pipelines of such ML operators over Spark. Complementary to these ideas, Hamlet [65] applied statistical learning theory to exploit certain database dependencies to drop features before ML without affecting accuracy significantly.

4.2 Model Selection and Management

Model selection is the overarching process of obtaining satisfactory ML models; it subsumes feature engineering and includes algorithm selection (AS) and hyper-parameter tuning (HT) [45]. It is typically an iterative human-in-the-loop process but most existing ML systems provide little support for optimizing this process end-to-end. Longview [6] proposed integrating "model management" functionality into a DBMS to automate certain aspects of model selection, along with managing learned ML models. AutoWeka [96] and MLbase [60] automate AS and HT, with MLbase's TUPAQ applying multi-armed bandit-style techniques [91]. Hemingway [77] automates AS and cluster size tuning jointly for distributed optimization algorithms, while [82] applied online aggregation-style techniques for HT in optimization algorithms. More generally, [64] outlined a vision of "model selection management systems" that build upon existing systems and combine ideas from the DB and ML to enable a wider spectrum of automation. Example systems include Sherlock [97], which proposed abstractions for iterative model building, ModelHub [71], which proposed a language and storage manager for managing deep neural networks that arise in computer vision, and ModelDB [98], which instruments ML libraries to capture and manage models.

Complementary to the above systems, cloud ML services, such as AzureML [2], aim at simple construction, scaling, and management of end-to-end ML workflows. A few recent systems target data cleaning and interactive specification as well. ActiveClean [61] integrates iterative data cleaning with SGD-based learning of convex ML models and proposes new sampling mechanisms to preserve convergence guarantees. Ava [53] provides a chat-bot front-end to make it easier to build models by maintaining a repository of "storyboards" for typical templates, while Vizdom [30] enables users to visually specify ML tasks on a touchscreen with incremental refinement of partial results. Finally, some systems aim to improve the usability and performance of serving trained ML models and predictions. Examples include Velox [27] and its successor Clipper [28], which enables model deployment across multiple frameworks, as well as MacroBase [12], which combines stream mining with streaming data explanation.

5. OPEN PROBLEMS

There are still various open research problems that need the attention of our community. We outline several problems although by no means is this an exhaustive list.

Size and Sparsity Estimation: Many optimization techniques require prior knowledge of the size and sparsity of matrices for cost comparisons and valid plan generation. But it is often non-trivial to infer such information for intermediates in complex linear algebra programs with conditional control flow, complex function call patterns, UDFs, and data-dependent operations. Hence, principled techniques for estimating the size and sparsity would be beneficial.

Convergence Estimation: Iterative ML algorithms often use convergence-based termination conditions. This leads to an unpredictable number of iterations, which makes it challenging to estimate the runtime, say, for resource allocation, or for decisions on expensive data re-organizations. Techniques to predict the number of iterations for convergence of ML algorithms would not only enable such optimizations but also enable progress estimators.

Adaptive Query Processing and Storage: Unknown or changing workloads are typically handled with adaptive query processing and storage techniques. But such concerns have received little attention in the context of large-scale ML. The state of the art is limited to inter-DAG dynamic recompilation [16] and lazy expression optimization [86].

Automatic Rewrites and Operator Fusion: Existing systems mostly apply coarse-grained pattern transformations, which limits rewrite and fusion potential. Rewrite frameworks would further benefit from a better exploitation of data flow (e.g., partitioning) and structural properties (e.g., diagonal matrices). Also, the increasing use of compression and new access methods make operator fusion far more challenging than on regular dense matrices.

Special Value Handling: Most systems ignore the effects of special values such as NaN or INF, which render, for example, sparse linear algebra invalid because $0 \cdot \text{NaN} = \text{NaN}$. The challenge is to support these special values during optimization and runtime without sacrificing performance.

Integrating Relational and Linear Algebra: A grand challenge is a seamless integration of relational and linear algebra so that users can specify, optimize, and execute ML tasks in a holistic framework, including data transformations for feature engineering such as joins and aggregates, and training and prediction of different ML models, including tasks such as cross-validation and feature selection.

Seamless Feature Engineering and Model Selection: To simplify end-to-end ML applications, there is a push towards (semi-)automating feature engineering and model selection [64]. Open questions include abstractions, meta-algorithms, and system architectures for different classes of ML models, including deep learning, which combines feature engineering and learning. An architecture that

leverages the progress on individual ML systems could be a feasible and impactful option. Other open questions involve applying and adapting theoretical results from learning theory, optimization, and human-computer interaction.

ML System Benchmarks: Existing benchmarks for large-scale analytics such as HiBench [51], BigBench [13], and SparkBench [5] contain a few ML tasks but only refer to reference implementations of large-scale ML libraries, which makes it hard to compare existing ML systems. There are also existing SQL-centric benchmarks for array databases [95] and Bayesian ML [20], but a broader range of benchmarks would be invaluable for the community at large.

6. BIOGRAPHIES

Arun Kumar is an Assistant Professor at the University of California, San Diego. He received his Ph.D. from the University of Wisconsin-Madison in 2016. His research interests are in the intersection of data management and ML, with a focus on making ML-based data analytics easier, faster, and more scalable. Ideas from his work have been adopted by many companies, including EMC, Oracle, Cloudera, and Facebook. He is a recipient of the Best Paper Award at SIGMOD 2014, the 2016 CS dissertation research award from UW-Madison, and a 2016 Google Faculty Research Award.

Matthias Boehm is a Research Staff Member at IBM Research – Almaden, where he is working since 2012 on optimization and runtime techniques for declarative, large-scale machine learning in SystemML. He received his Ph.D. from Technische Universitaet Dresden in 2011 with a dissertation on cost-based optimization of integration flows under the supervision of Prof. Wolfgang Lehner. His previous research also includes systems support for time series forecasting as well as in-memory indexing and query processing. In 2016, he received the VLDB Best Paper Award.

Jun Yang is a Professor of Computer Science at Duke University, where he has been teaching since receiving his Ph.D. from Stanford University in 2001. He is broadly interested in databases and data-intensive systems. He has been a recipient of the NSF CAREER Award, IBM Faculty Award, HP Labs Innovation Research Award, and Google Faculty Research Award. He also received the David and Janet Vaughan Brooks Teaching Award at Duke. His current research interests lie in making data analysis easier and more scalable for scientists, statisticians, and journalists.

7. REFERENCES

[1] Apache Mahout. mahout.apache.org.
[2] Microsoft AzureML Studio. studio.azureml.net.
[3] Oracle R Enterprise. www.oracle.com/technetwork/database/database-technologies/r/r-enterprise.
[4] M. Abadi et al. TensorFlow: A System for Large-Scale Machine Learning. *OSDI*, 2016.
[5] D. Agrawal et al. SparkBench - A Spark Performance Testing Suite. In *TPCTC*, 2015.
[6] M. Akdere et al. The Case for Predictive Database Systems: Opportunities and Challenges. In *CIDR*, 2011.
[7] A. Alexandrov et al. Implicit Parallelism through Deep Language Embedding. In *SIGMOD*, 2015.
[8] M. Anderson and M. Cafarella. Input Selection for Fast Feature Engineering. In *ICDE*, 2016.
[9] M. Anderson et al. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.
[10] M. Armbrust et al. Introduction to spark 2.0 for database researchers. In *SIGMOD*, 2016.
[11] A. Ashari et al. On Optimizing Machine Learning Workloads via Kernel Fusion. In *PPoPP*, 2015.
[12] P. Bailis et al. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD*, 2017.
[13] C. K. Baru et al. Discussion of BigBench: A Proposed Industry Standard Performance Benchmark for Big Data. In *TPCTC*, 2014.
[14] J. Bergstra et al. Theano: a CPU and GPU Math Expression Compiler. In *SciPy*, 2010.
[15] M. Boehm et al. Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML. *PVLDB*, 7(7), 2014.
[16] M. Boehm et al. SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. *IEEE Data Eng. Bull.*, 37(3), 2014.
[17] M. Boehm et al. SystemML: Declarative Machine Learning on Spark. *PVLDB*, 9(13), 2016.
[18] V. R. Borkar et al. Declarative Systems for Large-Scale Machine Learning. *IEEE Data Eng. Bull.*, 35(2), 2012.
[19] Z. Cai et al. Simulation of Database-valued Markov Chains Using SimSQL. In *SIGMOD*, 2013.
[20] Z. Cai et al. A Comparison of Platforms for Implementing and Running Very Large Scale Machine Learning Algorithms. In *SIGMOD*, 2014.
[21] L. Chen et al. Towards Linear Algebra over Normalized Data. *CoRR*, abs/1612.07448, 2016.
[22] Y. Cheng et al. GLADE: Big Data Analytics Made Easy. In *SIGMOD*, 2012.
[23] B.-G. Chun et al. Dolphin: Runtime Optimization for Distributed Machine Learning. In *ICML Workshop MLSystems*, 2016.
[24] J. Cohen et al. MAD Skills: New Analysis Practices for Big Data. *PVLDB*, 2(2), 2009.
[25] R. Collobert et al. Torch7: A Matlab-like Environment for Machine Learning. In *NIPS Workshop BigLearn*, 2011.
[26] T. Condie et al. Machine Learning for Big Data. In *SIGMOD*, 2013.
[27] D. Crankshaw et al. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *CIDR*, 2015.
[28] D. Crankshaw et al. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI*, 2017.
[29] A. Crotty et al. An Architecture for Compiling UDF-centric Workflows. *PVLDB*, 8(12), 2015.
[30] A. Crotty et al. Vizdom: Interactive Analytics through Pen and Touch. *PVLDB*, 8(12), 2015.
[31] C. De Sa et al. DeepDive: Declarative Knowledge Base Construction. *SIGMOD Record*, 45(1), 2016.
[32] A. Deshpande and S. Madden. MauveDB: Supporting Model-based User Views in Database Systems. In *SIGMOD*, 2006.
[33] P. Domingos. A Few Useful Things to Know About Machine Learning. *ACM CACM*, 55(10), 2012.
[34] S. Duan and S. Babu. Processing Forecasting Queries. In *VLDB*, 2007.
[35] T. Elgamal et al. SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. *CIDR*, 2017.
[36] A. Elgohary et al. Compressed Linear Algebra for Large-Scale Machine Learning. *PVLDB*, 9(12), 2016.
[37] S. Ewen et al. Spinning Fast Iterative Data Flows. *PVLDB*, 5(11), 2012.
[38] X. Feng et al. Towards a Unified Architecture for in-RDBMS Analytics. In *SIGMOD*, 2012.
[39] U. Fischer. *Forecasting in Database Systems*. PhD thesis, Technische Universitaet Dresden, 2014.
[40] Z. J. Gao et al. The BUDS Language for Distributed Bayesian Machine Learning. In *SIGMOD*, 2017.
[41] T. Ge and S. B. Zdonik. A Skip-list Approach for Efficiently Processing Forecasting Queries. *PVLDB*, 1(1), 2008.
[42] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.
[43] J. E. Gonzalez et al. Asynchronous Complex Analytics in a Distributed Dataflow Architecture. *CoRR*, 2015.
[44] I. Guyon et al. *Feature Extraction: Foundations and Applications*. New York: Springer-Verlag, 2001.
[45] T. Hastie et al. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer-Verlag, 2001.

[46] J. M. Hellerstein et al. The MADlib Analytics Library or MAD Skills, the SQL. *PVLDB*, 5(12), 2012.

[47] H. Herodotou et al. Starfish: A Self-tuning System for Big Data Analytics. In *CIDR*, 2011.

[48] B. Huang et al. Cumulon: Optimizing Statistical Data Analysis in the Cloud. In *SIGMOD*, 2013.

[49] B. Huang et al. Cumulon: Matrix-Based Data Analytics in the Cloud with Spot Instances. *PVLDB*, 9(3), 2015.

[50] B. Huang et al. Resource Elasticity for Large-Scale Machine Learning. In *SIGMOD*, 2015.

[51] S. Huang et al. The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In *ICDE Workshop WISS*, 2010.

[52] Y. Jia et al. Caffe: Convolutional Architecture for Fast Feature Embedding. *CoRR*, 2014.

[53] R. J. L. John et al. Ava: From Data to Insights Through Conversation. In *CIDR*, 2017.

[54] D. Kernert et al. SLACID - Sparse Linear Algebra in a Column-Oriented In-Memory Database System. In *SSDBM*, 2014.

[55] D. Kernert et al. SpMacho - Optimizing Sparse Linear Algebra Expressions with Probabilistic Density Estimation. In *EDBT*, 2015.

[56] D. Kernert et al. Topology-Aware Optimization of Big Sparse Matrices and Matrix Multiplications on Main-Memory Systems. In *ICDE*, 2016.

[57] M. A. Khamis et al. FAQ: Questions Asked Frequently. In *PODS*, 2016.

[58] M. Kim. *TensorDB and Tensor-Relational Model (TRM) for Efficient Tensor-Relational Operations*. PhD thesis, Arizona State University, 2014.

[59] M. L. Koc and C. Ré. Incrementally maintaining classification using an RDBMS. *PVLDB*, 4(5), 2011.

[60] T. Kraska et al. MLbase: A Distributed Machine-learning System. In *CIDR*, 2013.

[61] S. Krishnan et al. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *PVLDB*, 9(12), 2016.

[62] A. Kumar et al. Demonstration of Santoku: Optimizing Machine Learning over Normalized Data. *PVLDB*, 8(12), 2015.

[63] A. Kumar et al. Learning Generalized Linear Models Over Normalized Data. In *SIGMOD*, 2015.

[64] A. Kumar et al. Model Selection Management Systems: The Next Frontier of Advanced Analytics. *SIGMOD Record*, 2015.

[65] A. Kumar et al. To Join or Not to Join? Thinking Twice about Joins before Feature Selection. In *SIGMOD*, 2016.

[66] Y. Lecun et al. Deep Learning. *Nature*, 521:436–444, 5 2015.

[67] Y. Low et al. Distributed GraphLab: A Framework for Machine Learning in the Cloud. *PVLDB*, 5(8), 2012.

[68] R. Marcus and O. Papaemmanouil. WiSeDB: A Learning-based Workload Management Advisor for Cloud Databases. *PVLDB*, 9(10), 2016.

[69] C. Mayfield et al. ERACER: A Database Approach for Statistical Inference and Data Cleaning. In *SIGMOD*, 2010.

[70] X. Meng et al. MLlib: Machine Learning in Apache Spark. *JMLR*, 17(1), 2016.

[71] H. Miao et al. ModelHub: Towards Unified Data and Lifecycle Management for Deep Learning. *CoRR*, 2016.

[72] S. Narayanamurthy et al. Towards Resource-Elastic Machine Learning. In *NIPS Workshop BigLearn*, 2013.

[73] M. Nikolic et al. LINVIEW: Incremental View Maintenance for Complex Analytical Queries. In *SIGMOD*, 2014.

[74] F. Niu et al. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *NIPS*, 2011.

[75] F. Niu et al. Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS. *PVLDB*, 4(6), 2011.

[76] C. Ordonez et al. One-pass Data Mining Algorithms in a DBMS with UDFs. In *SIGMOD*, 2011.

[77] X. Pan et al. Hemingway: Modeling Distributed Optimization Algorithms. In *NIPS Workshop MLSystems*, 2016.

[78] S. Papadopoulos et al. The TileDB Array Data Storage Manager. *PVLDB*, 10(4), 2016.

[79] A. Pavlo et al. On Predictive Modeling for Optimizing Transaction Execution in Parallel OLTP Systems. *PVLDB*, 5(2), 2011.

[80] A. Pavlo et al. Self-Driving Database Management Systems. In *CIDR*, 2017.

[81] S. Prasad et al. Large-scale Predictive Analytics in Vertica: Fast Data Transfer, Distributed Model Creation, and In-database Prediction. In *SIGMOD*, 2015.

[82] C. Qin and F. Rusu. Speculative Approximations for Terascale Distributed Gradient Descent Optimization. In *SIGMOD DanaC*, 2012.

[83] C. Ré et al. Feature Engineering for Knowledge Base Construction. *IEEE Data Eng. Bull.*, 2014.

[84] S. Rendle. Scaling Factorization Machines to Relational Data. *PVLDB*, 6(5), 2013.

[85] S. Schelter et al. "All Roads Lead to Rome:" Optimistic Recovery for Distributed Iterative Data Processing. In *CIKM*, 2013.

[86] S. Schelter et al. Samsara: Declarative Machine Learning on Distributed Dataflow Systems. *NIPS Workshop MLSystems*, 2016.

[87] M. Schleich et al. Learning Linear Regression Models over Factorized Joins. In *SIGMOD*, 2016.

[88] J. Shin et al. Incremental Knowledge Base Construction Using DeepDive. *PVLDB*, 8(11), 2015.

[89] P. Shivam et al. Active and Accelerated Learning of Cost Models for Optimizing Scientific Applications. In *VLDB*, 2006.

[90] E. Soroush et al. ArrayStore: A Storage Manager for Complex Parallel Array Processing. In *SIGMOD*, 2011.

[91] E. R. Sparks et al. Automating Model Search for Large Scale Machine Learning. In *SoCC*, 2015.

[92] E. R. Sparks et al. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. In *ICDE*, 2017.

[93] M. Stonebraker et al. The Architecture of SciDB. In *SSDBM*, 2011.

[94] A. K. Sujeeth et al. OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning. In *ICML*, 2011.

[95] R. Taft et al. GenBase: A Complex Analytics Genomics Benchmark. In *SIGMOD*, 2014.

[96] C. Thornton et al. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *KDD*, 2013.

[97] M. Vartak et al. Supporting Fast Iteration in Model Building. In *NIPS Workshop LearningSys*, 2015.

[98] M. Vartak et al. MODELDB: A System for Machine Learning Model Management. In *SIGMOD Workshop HILDA*, 2016.

[99] S. Venkataraman et al. SparkR: Scaling R Programs with Spark. In *SIGMOD*, 2016.

[100] H. Wang et al. ATLAS: A Small but Complete SQL Extension for Data Mining and Data Streams. In *VLDB*, 2003.

[101] W. Wang et al. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Record*, 45(2), 2016.

[102] F. Wolf et al. Extending Database Task Schedulers for Multi-threaded Application Code. In *SSDBM*, 2015.

[103] D. Yan et al. Big Graph Analytics Systems. In *SIGMOD*, 2016.

[104] O. D. L. Yejas et al. Big R: Large-Scale Analytics on Hadoop Using R. In *Big Data*, 2014.

[105] C. Zhang et al. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*, 2014.

[106] C. Zhang and C. Ré. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In *SIGMOD*, 2013.

[107] C. Zhang and C. Ré. DimmWitted: A Study of Main-Memory Statistical Analytics. *PVLDB*, 7(12), 2014.

[108] M. Zhang et al. Measuring and Optimizing Distributed Array Programs. *PVLDB*, 9(12), 2016.

[109] Y. Zhang et al. RIOT: I/O-Efficient Numerical Computing without SQL. In *CIDR*, 2009.

[110] Y. Zhang et al. Storing Matrices on Disk: Theory and Practice Revisited. *PVLDB*, 4(11), 2011.

[111] Y. Zhang and J. Yang. Optimizing I/O for Big Array Analytics. *PVLDB*, 5(8), 2012.