Model Selection Management Systems: The Next Frontier of Advanced Analytics

Arun Kumar† Robert McCann‡ Jeffrey Naughton† Jignesh M. Patel†

†University of Wisconsin-Madison ‡Microsoft

†{arun, naughton, jignesh}@cs.wisc.edu, ‡robert.mccann@microsoft.com

ABSTRACT

Advanced analytics is a booming area in both industry and academia. Several projects aim to implement machine learning (ML) algorithms efficiently. But three key challenging and iterative practical tasks in using ML – *feature engineering*, *algorithm selection*, and *parameter tuning*, collectively called *model selection* – have largely been overlooked by the data management community, even though these are often the most time-consuming tasks for analysts. To make the iterative process of model selection easier and faster, we envision a unifying abstract framework that acts as a basis for a new class of analytics systems that we call *model selection management systems* (MSMS). We discuss how time-tested ideas from database research offer new avenues to improving model selection, and outline how MSMSs are a new frontier for interesting and impactful data management research.

1. INTRODUCTION

The data management community has produced successful systems that implement machine learning (ML) techniques efficiently, often over data management platforms [2, 6, 8, 11, 19]. But the process of building ML models for data applications is seldom a one-shot "slam dunk." Analysts face major practical bottlenecks in *using* ML that slow down the analytics lifecycle [3]. To understand these bottlenecks, we spoke with analysts at several enterprise and Web companies. Unanimously, they mentioned that choosing the right *features* and appropriately tuned ML models were among their top concerns. Other recent studies have produced similar findings [4, 5, 12]. In this paper, we focus on a related set of challenging practical tasks in using ML for data-driven applications: *feature engineering* (FE), in which the analyst chooses the features to use; *algorithm selection* (AS), in which the analyst picks an ML algorithm; and *parameter tuning* (PT), in which the analyst tunes ML algorithm parameters. These tasks, collectively called *model selection*, lie at the heart of advanced analytics.

Model Selection. Broadly defined, model selection is the process of building a precise prediction function to make "satisfactorily" accurate predictions about a data-generating process using data generated by the same process [10]. In this paper, we explain how viewing model selection from a data management standpoint can improve the process. To this end, we envision a *unifying framework* that lays a foundation for a new class of analytics systems: *model selection management systems.*

Model Selection Triple. A large body of work in ML focuses on various theoretical aspects of model selection [10]. But from a practical perspective, we found that analysts typically use an *iterative exploratory process.* While the process varies across analysts, we observed that the core object of their exploration is identical – an object we call the *model selection triple* (MST). It has three components: an FE "option" (loosely defined, a sequence of computation operations) that fixes the feature set that represents the data, an AS option that fixes the ML algorithm, and a PT option that fixes the parameter choices conditioned on the AS option. Model selection is iterative and exploratory because the space of MSTs is usually infinite, and it is generally impossible for analysts to know a priori which MST will yield satisfactory accuracy and/or insights.

Three-Phase Iteration. We divide an iteration into three phases, as shown in Figure 1(A). (1) *Steering*: the analyst decides on an MST and specifies it in an ML-related language or GUI such as R, Scala, SAS, or Weka. For example, suppose she has structured data; she might decide to use all features (FE option), and build a decision tree (AS option) with

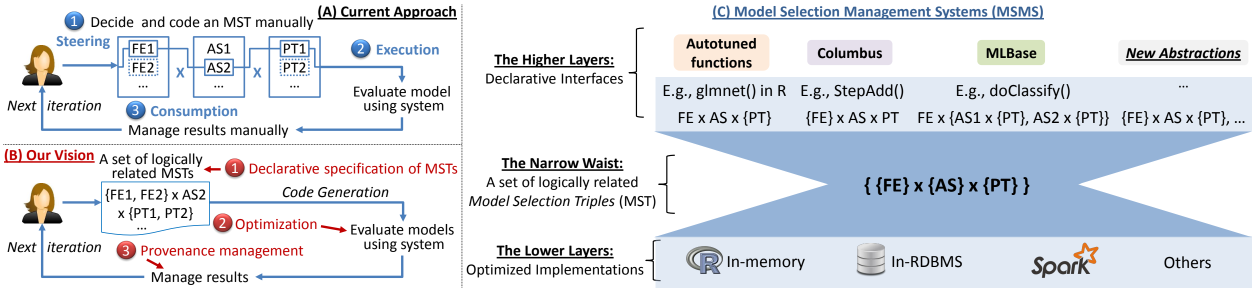


Figure 1: Model Selection. (A) The current dominant approach: the analyst chooses one combination of options for feature engineering (FE), algorithm selection (AS), and parameter tuning (PT); we call it a Model Selection Triple (MST). She iterates by modifying the MST, e.g., altering a parameter, or dropping a feature. (B) Our vision: she groups *logically related* MSTs, while the system optimizes the computations and helps manage results across iterations. (C) MSTs act as a unifying abstraction (a "narrow waist") for a new class of analytics systems that we call Model Selection Management Systems (MSMS).

a fixed tree height (PT option). (2) *Execution*: the system executes the MST to build and evaluate the ML model, typically on top of a data management platform, e.g., an RDBMS or Spark. (3) *Consumption*: the analyst assesses the results to decide upon the MST for the next iteration, or stops the process. For example, if the tree is too big, she might reduce the height (PT option changed), or drop some features (FE option changed). Even such minor changes in MSTs can cause major changes in accuracy and interpretability, and it is generally hard to anticipate such effects. Thus, analysts evaluate several MSTs using an iterative process.

Alas, most existing ML systems (e.g., [2, 6, 8, 11, 19]) force analysts to explore one MST per iteration. This overburdens the analyst, since she has to perform more iterations. Also, since the system is ignorant of the relationship between the MSTs explored, opportunities to speed up Execution are lost, while Consumption becomes more manual, which causes more pain for analysts. *Our vision aims to mitigate these issues by providing more systems support to improve the effectiveness of analysts as well as the efficiency of the iterative process of model selection.*

Automation Spectrum. A natural first step is to automate some of the analyst's work by providing a more holistic view of the process to the system. For example, the system can evaluate decision trees with all possible heights (PT options), or all subsets of features (FE options). Clearly, such naive brute-force automation might be prohibitively expensive, while the analyst's expertise might be wastefully ignored. On the other hand, if the system hardcodes only a few sets of MSTs, analysts might deem it too restrictive to be useful. Ideally, what we want is the flexibility to cover a wide *spectrum of automation*, in which the analyst controls exactly how much automation she desires for her application. This would enable analysts to still use their expertise during Steering, but push much of the "heavy lifting" to the system during Execution and Consumption.

1.1 Our Vision: MSMS to Manage MSTs

We envision a unifying framework that enables analysts to easily explore *a set of logically related MSTs per iteration* rather than just one MST per iteration. The analyst's expertise is useful in deciding which MSTs are grouped. Figure 1(B) illustrates our vision. In the figure, the analyst groups multiple values of tree height and feature subsets. As we will explain shortly, this ability to handle a logically related set of MSTs all at once is a simple but powerful unifying abstraction for a new class of analytics systems that aim to support the iterative process of model selection. We call such systems *model selection management systems* (MSMS).

Iteration in an MSMS. MSTs are too low-level for analysts to enumerate. Thus, an MSMS should provide a higher level of abstraction for specifying MSTs. A trivial way is to use *for* loops. But our vision goes deeper to exploit the full potential of our idea, by drawing inspiration from the RDBMS philosophy of handling "queries." By repurposing three key ideas from the database literature, an MSMS can make model selection significantly easier and faster. (1) Steering: an MSMS should offer a *framework of declarative operations* that enable analysts to easily group logically related MSTs. For example, the analyst can just "declare" the set of tree heights and feature subsets (projections). The system generates lower-level code to implicitly enumerate the MSTs encoded by the declarative operations. (2) Execution: an MSMS should include *optimization* techniques to reduce the runtime per iteration by exploiting the set-oriented nature of specifying MSTs. For example, the system could share computations across different parameters or share intermediate materialized data for different feature sets. (3) Consumption: an MSMS should offer *provenance management* so that the system can help the analyst manage results and help with optimization. For example, the analyst can inspect the results using standard queries to help steer the next iteration, while the system can track intermediate data and models for reuse. *Overall, an MSMS that is designed based on our unifying framework can reduce both the number of iterations (by improving Steering and Consumption) and the time per iteration (by improving Execution).*

Unifying Narrow Waist. Interestingly, our framework subsumes many prior abstractions that can be seen as basically restricted MSMS interfaces in hindsight, as shown in Figure 1(C). In a sense, our abstraction acts as a "narrow waist" for MSMSs.[1] An MSMS stacks higher layers of abstraction (declarative interfaces) to make it *easier to specify sets of MSTs* and lower layers of optimized implementations to *exploit the set-oriented nature of specifying MSTs*. We elaborate with three key examples: (1) R provides many autotuned functions, e.g., *glmnet* for linear models. These can be viewed as declarative operations to explore multiple PT options, while fixing FE and AS options. (2) Columbus [18] offers declarative operations to explore a set of feature subsets simultaneously. This can be viewed as exploring multiple FE options, while fixing AS and PT options. (3) MLBase [13] fully automates algorithm selection and parameter tuning by hardcoding a small set of ML techniques and tuning heuristics (unlike our vision of handling a wide spectrum of automation). This can be viewed as exploring multiple combinations of AS and PT options, while fixing the FE option. *Our vision is the distillation of the common thread across such abstractions, and lays a principled foundation for the design and evaluation of model selection management systems.*

Towards a Unified MSMS. The natural next step is to build a unified MSMS that exposes the full power of our abstraction rather than supporting FE, AS, and PT in a piecemeal fashion. A unified MSMS could make it easier for analysts to handle the whole process in one "program" rather than straddling multiple tools. It also enables sharing code for tasks such as cross-validation, which is needed for each of FE, AS, and PT. However, building a unified MSMS poses research challenges that data management researchers are perhaps more familiar with than ML researchers, e.g., the design trade-offs for declarative languages. A unified MSMS also requires ideas from data management and ML, e.g., materializing intermediate data, or sharing computations, which requires RDBMS-style cost-based analyses of ML algorithms. The data management community's expertise in designing query optimizers could be useful here. A caveat is that the three tasks, especially FE, involve a wide variety of operations. It is perhaps infeasible to capture all such operations in the first design of a unified MSMS. Thus, any such MSMS must be extensible. Next, we provide more background on FE, AS, and PT.

2. MORE BACKGROUND

Feature Engineering (FE) is the process of converting raw data into a precise feature vector that provides the domain of the prediction function (a learned ML model) [5]. FE includes a variety of *options* (a sequence of computational operations), e.g., counting words or selecting a feature subset. Some options, such as subset selection and feature ranking, are well studied [9]. FE is considered a domain-specific "black art" [4, 5], mostly because it is influenced by many technical and logistical factors, e.g., data and application properties, accuracy, time, interpretability, and company policies. Unfortunately, there is not much integrated systems support for FE, which often forces analysts to write scripts in languages external to data management systems, sample and migrate data, create intermediate data, and track their steps manually [4, 12]. Such manual effort slows and inhibits exploration.

Algorithm Selection (AS) is the process of picking an ML model, i.e., an *inductive bias*, that fixes the hypothesis space of prediction functions explored for a given application [10]. For example, logistic regression and decision trees are popular ML techniques for classification applications. Some ML models have multiple learning algorithms; for example, logistic regression can use both batch and stochastic gradient methods. Like FE, AS depends on both technical and non-technical factors, which leads to a combinatorial explosion of choices. Learning ensembles of ML models is also popular [10].

[1] Like how IP acts as the "narrow waist" of the Internet.

This complexity often forces analysts to iteratively try multiple ML techniques, which often leads to duplicated effort, and wasted time and resources.

Parameter Tuning (PT) is the process of choosing the values of (hyper-)parameters that many ML models and algorithms have. For example, logistic regression is typically used with a parameter known as the *regularizer*. Such parameters are important because they control accuracy-performance trade-offs, but tuning them is challenging partly because the optimization problems involved are usually non-convex [10]. Thus, analysts typically perform ad hoc manual tuning by iteratively picking a set of values, or by using heuristics such as *grid search* [10]. Some toolkits automate PT for popular ML techniques, which could indeed be useful for some applications. But from our conversations with analysts, we learned that they often tend to avoid such "black box" tuning in order to exercise more control over the accuracy-performance trade-offs.

3. RESEARCH CHALLENGES AND DESIGN TRADE-OFFS

We discuss the key challenges in realizing our vision of a unified MSMS and explain how they lead to novel and interesting research problems in data management. We also outline potential solution approaches, but in order to provide a broader perspective, we explain the design trade-offs involved rather than choosing specific approaches.

3.1 Steering: Declarative Interface

The first major challenge is to make it easier for analysts to specify sets of logically related MSTs using the power of declarative interface languages. There are two components to this challenge.

Language Environment: This component involves deciding whether to create a new language or embedded domain-specific languages (DSLs). The former offers more flexibility, but it might isolate us from popular language environments such as Python, R, and Scala. A related decision is whether to use logic or more general dataflow abstractions as the basis. The latter might be less rigorous but they are more flexible. The lessons of early MSMSs suggest that DSLs and dataflow abstractions are preferable; for example, Columbus provides a DSL for R [18]. The expertise of the data management community with declarative languages will be crucial here.

Scope and Extensibility: Identifying the right declarative primitives is a key challenge. Our goal is to capture a wide spectrum of automation. Thus, we need several predefined primitives to hardcode common operations for each of FE, AS, and PT as well as popular ways of combining MSTs. For example, for FE, standardization of features and joins are common operations, while subset selection is a common way of combining MSTs. For AS and PT, popular combinations and parameter search heuristics can be supported as first-class primitives, but we also need primitives that enable analysts to specify custom combinations based on their expertise. Of course, it is unlikely that one language can capture all ML models for AS or all operations for FE and PT. Moreover, different data types (structured data, text, etc.) need different operations. A pragmatic approach is to start with a set of most common and popular operations as first-class citizens that are optimized, and then expand systematically to include more. Thus, the language needs to be extensible, i.e., support user-defined functions, even if they are less optimizable.

3.2 Execution: Optimization

To fully exploit declarativity, an MSMS should use the relationship between MSTs to optimize the execution of each iteration. Faster iterations might encourage analysts to explore more MSTs, leading to more insights. This challenge has three aspects.

Avoiding Redundancy: Perhaps the most important and interesting aspect is to avoid redundancy in both data movement and computations, since the MSTs grouped together in one iteration are likely to differ only slightly. This idea has been studied before, but its full power is yet to be exploited, especially for arbitrary sets of MSTs. For example, Columbus [18] demonstrated a handful of optimizations for multiple feature sets being grouped together during subset selection over structured data. Extending it to other aspects of FE as well as to AS and PT is an open problem. For example, we need not build a decision tree from scratch for different height parameters, if monotonicity is ensured. Another example is sharing computations across different linear models. Redundancy can also be avoided within a single MST; for example, combining the FE option of joins with the AS option of learning a linear model could avoid costly denormalization [15] or even whole input tables [16]. Extending this to other ML models is an open question. Such optimizations require complex performance trade-offs involving data and system properties, which might be unfamiliar to ML researchers. Thus, the expertise of the data management community in designing cost models and optimizers is crucial here.

System Flexibility: This aspect relates to what lies beneath the declarative language. One approach is to build an MSMS on top of existing data platforms such as Spark, which might make adoption easier, but might make it more daunting to include optimizations that need changes to the system code. An alternative is to build mostly from scratch, which would offer more flexibility but requires more time and software engineering effort. This underscores the importance of optimizations that are generic and easily portable across data platforms.

Incorporating Approximation: Many ML models are robust to perturbations in the data and/or the learning algorithm, e.g., sampling or approximations. While such ideas have been studied for a single MST, new opportunities arise when multiple MSTs are executed together. For example, one could "warm start" models using models learned on a different feature subset [18]. A more challenging question is whether such warm starting is possible across different ML models. Finally, new and intuitive mechanisms to enable analysts to trade off time and accuracy can be studied by asking analysts to provide desired bounds on time or accuracy in the declarative interface. The system could alter its search space on the fly and provide interactive feedback. Exploiting such opportunities requires characterization of new accuracy-performance trade-offs, which might require the data management community to work more closely with ML researchers.

3.3 Consumption: Managing Provenance

Operating on more MSTs per iteration means the analyst needs to consume more results and track the effects of their choices more carefully. But thanks to declarativity, the system can offer more pervasive help for such tasks. This has two aspects.

Capture and Storage: The first aspect is to decide what to capture and how to store it. Storing information about all MSTs naively might cause unacceptable storage and performance overheads. For example, even simple subset selection operations for FE on a dataset with dozens of features might yield millions of MSTs. One approach is to design special provenance schemas based on the semantics of the declarative operations. Another approach is to design new compression techniques. The analyst might have a key role to play in deciding exactly what needs to be tracked; for example, they might not be interested in PT-related changes, but might want to inspect AS- and FE-related changes. Novel applications are possible if these problems are solved, e.g., auto-completion or recommendation of MST changes to help analysts improve Steering. The expertise of the data management community with managing workflow and data provenance could be helpful in tackling such problems. Building applications to improve analyst interaction using provenance might require more collaboration with the human-computer interaction (HCI) community.

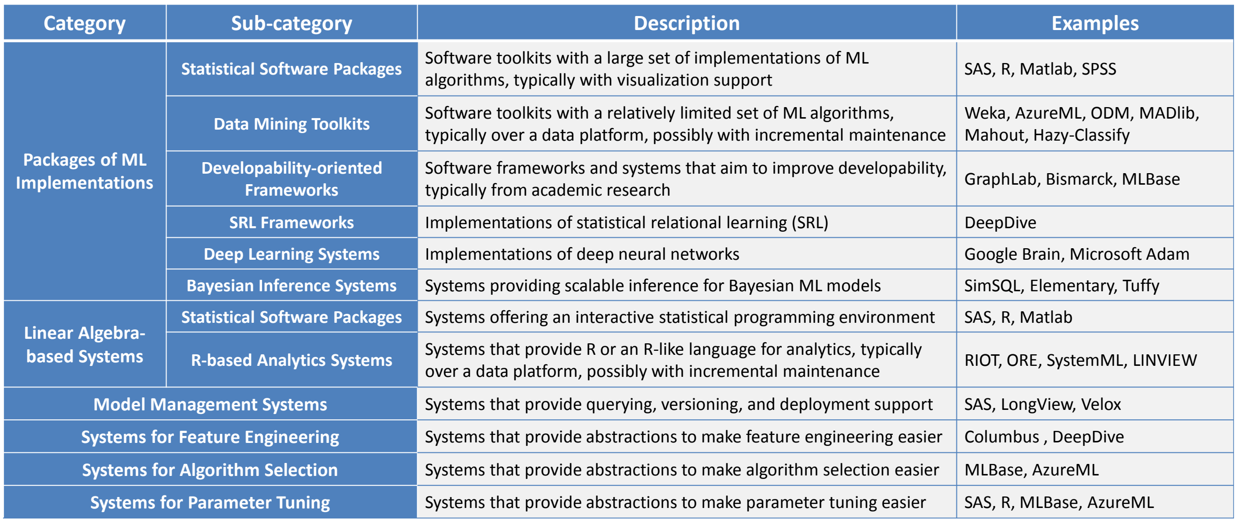
Reuse and Replay: Another aspect is the interaction of provenance with optimization. A key application is avoiding redundancy *across* iterations by reusing intermediate data and ML models. Such redundancy can arise, since MSTs typically differ only slightly across iterations, or if there are multiple analysts. This could involve classically hard problems such as relational query equivalence, but also new problems such as defining hierarchies of "subsumption" among ML models. For example, it is easy to reuse intermediate results for logistic regression if the number of iterations is increased by the analyst, but it is non-obvious to decide what to reuse if she drops some data examples or features. This points to the need to characterize a formal notion of "ML provenance," which is different from both data and workflow provenance. The data management community's expertise with formal provenance models could be helpful in tackling this challenge.

Summary. Building a unified MSMS to expose the full power of our abstraction requires tackling challenging research problems, which we outlined with potential solutions and design trade-offs. Our list is not comprehensive – other opportunities also exist, e.g., using visualization techniques to make Steering and Consumption easier. We hope to see more of such interesting new problems in MSMS research.

4. EXISTING LANDSCAPE

We now briefly survey the existing landscape of ML systems and discuss how our vision relates to them. We classify the systems into six categories based on their key goals and functionalities. Due to space constraints, we provide our survey in a separate report [14], but summarize it in Table 1. Our taxonomy is not intended to be exhaustive, but to give a picture of the gaps in the existing landscape.

Numerous systems have focused on efficient and/or scalable implementations of ML algorithms and/or R-like languages. Some other systems have focused



Category	Sub-category	Description	Examples
Packages of ML Implementations	Statistical Software Packages	Software toolkits with a large set of implementations of ML algorithms, typically with visualization support	SAS, R, Matlab, SPSS
	Data Mining Toolkits	Software toolkits with a relatively limited set of ML algorithms, typically over a data platform, possibly with incremental maintenance	Weka, AzureML, ODM, MADlib, Mahout, Hazy-Classify
	Developability-oriented Frameworks	Software frameworks and systems that aim to improve developability, typically from academic research	GraphLab, Bismarck, MLBase
	SRL Frameworks	Implementations of statistical relational learning (SRL)	DeepDive
	Deep Learning Systems	Implementations of deep neural networks	Google Brain, Microsoft Adam
	Bayesian Inference Systems	Systems providing scalable inference for Bayesian ML models	SimSQL, Elementary, Tuffy
Linear Algebra-based Systems	Statistical Software Packages	Systems offering an interactive statistical programming environment	SAS, R, Matlab
	R-based Analytics Systems	Systems that provide R or an R-like language for analytics, typically over a data platform, possibly with incremental maintenance	RIOT, ORE, SystemML, LINVIEW
Model Management Systems		Systems that provide querying, versioning, and deployment support	SAS, LongView, Velox
Systems for Feature Engineering		Systems that provide abstractions to make feature engineering easier	Columbus , DeepDive
Systems for Algorithm Selection		Systems that provide abstractions to make algorithm selection easier	MLBase, AzureML
Systems for Parameter Tuning		Systems that provide abstractions to make parameter tuning easier	SAS, R, MLBase, AzureML

Table 1: Major categories of ML systems surveyed, along with examples from both products and research. It is possible for a system to belong to more than one category, since it could have multiple key goals.

on "model management"-related issues, which involve logistical tasks such as deployment and versioning. A few recent systems aim to tackle one or more of FE, AS, and PT – Columbus, MLBase, DeepDive, and AzureML. However, they either do not abstract the whole process of model selection as we do, or do not aim to support a wide portion of the automation spectrum. We have already discussed Columbus and MLBase in Section 1. DeepDive provides a declarative language to specify factor graph models and aims to make FE easier [17], but it does not address AS and PT. Automation of PT using massive parallelism has also been studied [7]. AzureML provides something similar, and it also aims to make it easier to manage ML workflows for algorithm selection [1]. All these projects provided the inspiration for our vision. We distill their lessons as well as our interactions with analysts into a unifying abstract framework. We also take the logical next step of envisioning a unified MSMS based on our framework to support FE, AS, and PT in an integrated fashion (Figure 1).

5. CONCLUSION

We argue that it is time for the data management community to look beyond just implementing ML algorithms efficiently and help improve the iterative process of model selection, which lies at the heart of using ML for data applications. Our unifying abstraction of model selection triples acts as a basis for designing a new class of analytics systems to manage model selection in a holistic and integrated fashion. By leveraging three key ideas from data management research – declarativity, optimization, and provenance – such model selection management systems could help make model selection easier and faster. This could be a promising direction for interesting and impactful research in data management, as well as its intersection with ML and HCI.

6. REFERENCES

[1] Microsoft Azure ML. studio.azureml.net.
[2] Oracle R Enterprise. www.oracle.com.
[3] SAS Report on Analytics. sas.com/reg/wp/corp/23876.
[4] M. Anderson et al. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.
[5] P. Domingos. A Few Useful Things to Know about Machine Learning. *CACM*, 2012.
[6] X. Feng et al. Towards a Unified Architecture for in-RDBMS Analytics. In *SIGMOD*, 2012.
[7] Y. Ganjisaffar et al. Distributed Tuning of Machine Learning Algorithms Using MapReduce Clusters. In *LDMTA*, 2011.
[8] A. Ghoting et al. SystemML: Declarative Machine Learning on MapReduce. In *ICDE*, 2011.
[9] I. Guyon et al. *Feature Extraction: Foundations and Applications*. New York: Springer-Verlag, 2001.
[10] T. Hastie et al. *Elements of Statistical Learning: Data mining, inference, and prediction*. Springer-Verlag, 2001.
[11] J. Hellerstein et al. The MADlib Analytics Library or MAD Skills, the SQL. In *VLDB*, 2012.
[12] S. Kandel et al. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE TVCG*, 2012.
[13] T. Kraska et al. MLbase: A Distributed Machine-learning System. In *CIDR*, 2013.
[14] A. Kumar et al. A Survey of the Existing Landscape of ML Systems. *UW-Madison CS Tech. Rep. TR1827*, 2015.
[15] A. Kumar et al. Learning Generalized Linear Models Over Normalized Data. In *SIGMOD*, 2015.
[16] A. Kumar et al. To Join or Not to Join? Thinking Twice about Joins before Feature Selection. In *SIGMOD*, 2016.
[17] C. Ré et al. Feature Engineering for Knowledge Base Construction. *IEEE Data Engineering Bulletin*, 2014.
[18] C. Zhang et al. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*, 2014.
[19] Y. Zhang et al. I/O-Efficient Statistical Computing with RIOT. In *ICDE*, 2010.