A Survey of the Existing Landscape of ML Systems

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

We survey the existing landscape of ML systems to identify gaps that motivate our vision of a unifying abstraction to support the iterative process of *model selection* and lay a principled foundation for *model selection management systems*.

1. INTRODUCTION

We present a detailed survey of the existing landscape of ML systems. We categorize the existing and proposed ML systems into six major categories: (1) Packages of ML Implementations, (2) Systems with a Linear Algebra-based Language, (3) Model Management Systems, (4) Systems for Feature Engineering, (5) Systems for Algorithm Selection, and (6) Systems for Parameter Tuning. For each category (or sub-category, wherever applicable), we discuss a few prominent examples from both research and practice. Note that it is possible for a system to belong to more than one category, since it could potentially have multiple simultaneous goals. Our categorization is not intended to be exhaustive. Rather, we aim to give a high-level picture of the kinds of functionalities that have been considered, and underscore the gaps that exist to motivate our vision [26]. Table 1 summarizes the categories.

2. PACKAGES OF ML IMPLEMENTATIONS

Over the last two decades, both the industry and academic research projects have produced packages and toolkits that implement many ML algorithms as well as other statistical functionalities. We classify these systems loosely into three sub-categories.

2.1 Statistical Software Packages

These systems provide implementations of many established as well as new statistical and ML techniques. Commercial packages such as SAS and SPSS are used widely in enterprise settings and are often integral to their data analytics infrastructure [6]. Typically, these packages also provide rich visualization capabilities as well as graphical user-interfaces that help improve the *usability* of such systems, say, for analysts easily visualize the effects of their ML models [6]. R is a popular open-source environment and language that provides free implementations of almost all well-known ML techniques in the form of libraries. R has a large community of users that contribute the code of new techniques, including many domain-specific ones. [1] And similar to the commercial packages, R also provides a rich set of visualization capabilities as well as extensibility mechanisms to connect it to other languages and systems. These packages were developed primarily for in-memory and interactive usage, i.e., for scenarios in which the data fit in memory, and an analyst is in the loop with the system. However, some of them also provide scalable implementations of a subset of ML techniques.

2.2 Data Mining Toolkits

Unlike the statistical software packages, data mining toolkits have a narrower focus and provide a smaller set of ML implementations. The open-source toolkit Weka is popular for academic usage, and is primarily meant for in-memory and interactive usage. [2] Mahout is an open-source toolkit that provides implementations of some ML techniques on Hadoop [1]. Thus, Mahout is primarily meant for distributed and batch processing of large amounts of data on a cluster. Mahout, and similar distributed implementations (that are proprietary), are popular among Web companies that deal with PBs of data. Most database companies also sell their own data

[1]http://cran.r-project.org
[2]http://www.cs.waikato.ac.nz/ml/weka

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.

mining toolkits that implement a set of ML techniques in an in-RDBMS fashion, i.e., they operate directly over data resident in an RDBMS. These toolkits are used primarily by enterprise companies that use RDBMSs to manage their data. MADlib is an open-source project similar to Mahout, except that the ML techniques are implemented in an RDBMS instead of Hadoop [21]. Both the RDMBS-based and Hadoop-based implementations provide *scalability* for the ML techniques. AzureML provides a cloud-based visual environment for constructing ML workflows [2]. It provides scalable and parallel implementations of popular ML techniques as well as data processing capabilities. Other projects that provide scalable and/or usable implementations of ML techniques include Vowpal Wabbit[3]. Some systems also provide *incremental maintenance* of the learned ML models and/or inference results as the underlying datasets evolve. For example, the Hazy-Classify system includes several optimization techniques to incrementally maintain the inference results of linear classifiers in an RDBMS [23].

2.3 Developability-oriented Frameworks

Complementing the efforts of both industry and the open-source community, a number of research projects have also produced systems that provide implementations of ML techniques. However, rather than simply provide a "laundry list" of implementations, these projects aim to improve the productivity of the software developers that implement the ML techniques, i.e., the *developability* of ML systems. We provide a brief description a few major research projects in this space.

Graphlab provides a graph-based data model, and a vertex-oriented programming abstraction for implementing graph analysis and graph-based ML algorithms [27]. Examples of popular ML techniques that fit well into its abstraction include matrix completion using Alternating Least Squares, lasso using a form of coordinate descent, and Gaussian Mixture Models. By abstracting out graph-based computations into the three stages of Gather, Apply, and Scatter, GraphLab obviates the need for a developer to handle low-level implementation details such as scheduling, parallelization, and fault tolerance. In most cases, Graphlab requires the graph to fit in memory, but there are different versions for single-node and distributed memory. It uses an MPI-based communication framework in its implementation.

Bismarck provides a unified framework for integration of convex optimization-based ML techniques into an RDBMS by using stochastic gradient descent (SGD) as the underlying optimization algorithm [14]. Examples of popular ML techniques that fit their framework include logistic regression, linear regression and lasso, linear Support Vector Machines, and Conditional Random Fields. By exploiting the abstraction of a *user-defined aggregate function* that almost every major RDBMS provides, Bismarck obviates the need for a developer to han-

[3] `http://hunch.net/~vw`

dle low-level implementation details such as scalability to larger-than-memory data, and parallelization. Instead, it simply uses the underlying RDBMS for data management and parallelization. On shared-memory parallel systems, Bismarck uses the HOGWILD! approach of racing updates to parallelize SGD [31]. But on shared-nothing systems, including Hadoop-based systems such as Hive, it uses a model averaging approach for SGD [37].

The MLBase system aims to improve both the developability and usability of distributed machine learning [25] To improve developability, MLBase provides an API named MLI that is built around two data types – MLTable, which is basically a table, and LocalMatrix, which is simply a node-local matrix. MLI provides a library of interfaces as well as some implementations for a suite of common operations on these two data types that help in implementing common ML algorithms with a few lines of code. Examples of popular ML techniques that can be handled with their API include matrix completion using Alternating Least Squares as well as many convex optimization-based techniques using SGD (similar to Bismarck). Since MLBase uses Spark[4] as its underlying data processing engine, a developer need not worry about issues such as distributed execution and fault tolerance.

2.4 Statistical Relational Learning (SRL) Frameworks

Over the last decade, statistical relational learning models have increased in popularity [16]. Essentially, these sophisticated techniques enable one to learn a joint model over an entire database of relations with complex relationships among them. One of the most prominent of such models is called a Markov Logic Network (MLN).

An MLN is basically a set of first-order logic rules over a database schema, but each rule is assigned a numeric weight that captures the *uncertainty* of the rule being true [30]. Learning an MLN involves computing these weights given a training dataset. Inference with an MLN involves determining missing facts in an incomplete query relation (a "possible world" in probabilistic database terminology), possibly with their marginal probabilities. MLNs are a powerful framework that unify logical and statistical approaches to machine learning. In fact, many popular ML techniques can be expressed using MLNs, e.g., logistic regression, Conditional Random Fields, and correlation clustering.

Alchemy is the first-implementation of learning and inference in MLNs, but it is primarily an in-memory toolkit [4]. DeepDive is a more scalable system for statistical relational learning that is primarily designed for knowledge-base construction [33]. It generalizes an MLN in the sense that it enables an analyst to specify more general relationships than first-order logic rules by using user-defined functions in Python. DeepDive takes care of scalable learning of, and inference on, *factor graphs*, which is the underlying graph-based formalism for MLNs.

2.5 Deep Learning Systems

Neural networks with several hidden layers, known as "deep learning" networks, are increasing in popularity, especially for complex ML tasks such as speech recognition and computer vision. This is primarily due to the their ability to learn complex non-linear functions in an unsupervised manner as well as the decreasing cost of cluster computing.

Multiple Web companies have built their own deep learning systems, with Google publishing some technical details of their system [12]. Google's Brain system is based on a distributed learning framework to train a large, five-layer neural network on a cluster with hundreds of machines. They devise a new technique to parallelize SGD using a combination of model and data parallelism as well as asynchronous model updates. A similar parallelization technique is devised for L-BFGS. Their implementation of SGD achieves state-of-the-art quality and performance on benchmark tasks in speech and image recognition. Since neural networks obviate the need for engineering features, deep learning systems are increasingly used for such complex ML tasks, even though the learned models are mostly *uninterpretable* to humans [12]. Several other Web-related companies such as Baidu, Facebook, and Microsoft have also built their own deep learning systems.

2.6 Scalable Bayesian Inference Systems

A number of systems have focused on scaling and/or speeding up inference in complex Bayesian ML models and factor graphs using data management ideas. Inference in such models typically involves combinatorial problems, and often requires iterative sampling methods such as Gibbs sampling [28].

The Tuffy system aims to scale MLN inference using an RDBMS [30]. Inference in an ML involves two phases – grounding (loosely defined as assigning possible values to free random variables), and

[4] `https://spark.apache.org`

search (exploring different feasible combinations of variable assignments to minimize a cost function). Tuffy recognizes that grounding involves relational joins, and exploits the RDBMS join optimizer by expressing the grounding phase in SQL (unlike Alchemy, which uses a naive in-memory nested loops join [4]). Additionally, Tuffy implements a hybrid architecture that performs grounding using SQL, but search using a specialized in-memory subsystem that is faster. Hence, Tuffy achieves higher scalability than the alternative in-memory tools for MLN inference such as Alchemy. Additionally, Tuffy exploits special structures in the factor graphs underlying the MLNs to enable partitioned parallelism during the inference, which improves performance further.

The Elementary system, built as part of DeepDive, aims to provide high-throughput inference via Gibbs sampling on large factor graphs [35]. Elementary maintains a factor graph using separate relations for the random variables and the factors. It optimizes the core operation of Gibbs sampling-based inference by casting it as a relational join, and applying several data management techniques. However, since the variables are updated during the sampling, the process could involve random reads and writes to relations. Hence, Elementary considers alternative view materialization strategies that reduces I/O costs. Additionally, Elementary also considers alternative buffer replacement strategies and page layouts for the relations as part of its tradeoff space for optimization. Since Elementary explores more choices for these classical issues, it is faster than existing RDBMS-based as well as other systems for Gibbs sampling. Furthermore, since Elementary is designed as a middleware layer, it can be integrated into a file system, key-value stores, or other systems as well.

The SimSQL system aims to enable SQL-based specification, simulation, and querying of database-valued Markov Chains that is useful for Bayesian ML [10]. SimSQL handles random variables as database tables, and uses user-defined functions in SQL to sample from various statistical distributions. The generative process for a Bayesian ML model can be specified in an extension of SQL that includes recursive table definitions and iteration. Thus, SimSQL enables inference techniques such as Gibbs sampling, which is useful for Bayesian ML techniques such as Bayesian linear regression and lasso, Latent Dirichlet Allocation (LDA), and Gaussian Mixture Models (GMMs). SimSQL converts the model specifications into a logical query plan and optimizes them dynamically using partitioning and batching heuristics. The system translates all operations into MapReduce jobs executed with Hadoop. SimSQL also improves developability because Bayesian ML models can be expressed succintly in tens of lines of SQL code instead of hundreds of lines in a procedural language such as Java or C++.

3. SYSTEMS WITH A LINEAR ALGEBRA-BASED LANGUAGE

Many ML techniques can be elegantly expressed as algebraic computations over matrices. Examples of popular ML techniques that can be implemented using a linear algebra-based language include linear regression with conjugate gradient, logistic regression with trust region methods, and Gaussian non-negative matrix factorization with multiplicative updates.

3.1 Statistical Software Packages

Apart from libraries of ML implementations, statistical software packages such as R and SAS also include a linear algebra-based language [5,6]. The language is usually interpreted, and the packages provide an interactive console-like environment for analysts to write statistical scripts. Essentially, datasets can be loaded into the language environment and manipulated using the operations of the language as well as user-written functions. Furthermore, these toolkits include general programming language functionalities such as iterations and conditionals as well. Thus, unlike most of the other systems in our discussion, the languages provided by these toolkits are Turing-complete. Analysts can output desired results to text files, or other formats.

3.2 R-based Analytics Systems

Over the last decade, R has become one of the most popular environments for ad-hoc statistical computing [5]. However, one of the main drawbacks of R is that the data have to fit in memory. The RIOT project aimed to address this issue by abstracting out the data flow in R to enable *transparent* execution of R scripts on larger-than-memory data [36]. Essentially, RIOT provides a middleware that translates the operations in R into procedural SQL-based queries over an RDBMS. RIOT also identifies many opportunities for optimizing the performance of the system during the translation. Many database companies such as EMC, HP, Oracle, and SAP have created products with similar goals as RIOT. For example, Oracle R Enterprise

provides R as a front-end for an analyst and translates the computations over R data frames transparently to queries over an RDMBS (or Hive) backend [3]. In contrast, IBM's SystemML project departs from R's interpreted environment and takes a compilation-based approach [17]. SystemML provides its own R-like language (but not entire R per se) called DML. Statistical programs written in DML are compiled down to MapReduce jobs on Hadoop. Akin to a classical RDBMS optimizing a SQL query, SystemML performs a number of optimizations while translating in order to improve overall performance.

LINVIEW is a middleware for R-based analytics systems that incorporates incremental maintenance techniques [29]. By maintaining "delta" matrices that capture small changes to input matrices efficiently, and by propagating them through linear algebra programs cleverly, LINVIEW improves the overall performance for small (low-rank) changes in the input matrices. The system includes a compiler that automatically identifies the opportunities for incremental updates on a given linear algebra-based program that is based on common matrix operations including matrix multiplication and matrix inversion. LINVIEW achieves quadratic time complexity for incrementally updating matrix multiplication – an operation that would otherwise have near-cubic complexity. But the tradeoff is that LINVIEW has a significantly higher memory footprint than re-evaluation, since it needs to maintain multiple intermediate matrices in memory. Overall, LINVIEW provides incremental maintenance for popular ML techniques based on linear algebra such as ordinary least squares linear regression, and some batch gradient descent-based methods.

4. MODEL MANAGEMENT SYSTEMS

Apart from standard learning and inference tasks, a few systems aim to address auxiliary "model management" tasks that involve managing metadata at different stages of the analytics lifecycle. Examples of such tasks include integrated and ad-hoc querying of ML models along with the data, versioning and tracking the evolution of ML models, and deployment of ML models to production. Many commercial data mining toolkits integrated into RDBMSes (say, from Oracle, Microsoft, or IBM) as well as statistical software packages such as SAS also provide some model management capabilities. In order to avoid repeating the discussion of those systems, we restrict this discussion to systems that addressed or raised interesting research questions.

4.1 LongView

The LongView project envisions a system that enables integrated management and querying of relational data and ML models in an RDBMS [7]. The authors draw high-level parallels between the use of declarative query languages and cost-based optimization in RDBMSs and the process of building and using ML models. They extend SQL with a simple API that captures the basic tasks in an applied ML workflow such as learning an ML model and using it for prediction. The system is expected to automatically train a specified ML model by handling parameter tuning, feature selection, and sampling. The authors mention the possibilities of new optimization techniques such as sharing work across models. Model metadata such as parameters used, training time, accuracy, etc. are stored and managed as tables. However, it is not clear if a user can understand or steer exactly how the system builds the ML models, especially how it handles parameter tuning, and feature selection.

4.2 Velox

The Velox project aims to make it easier to deploy learned ML models into production for online applications [11]. It includes a distributed memory-based engine that caches both the models and predictions over data for low-latency delivery of predictions to user-facing applications such as recommendation systems. Velox also handles new data examples, e.g., new ratings by users, by retraining the learned ML models incrementally using a new heuristic for matrix completion. In order avoid the ML models from becoming stale, Velox includes techniques to automatically monitor the quality of the predictions and trigger offline retraining of ML models using Spark and MLBase [25]. Velox also includes techniques for diversifying the predictions in order to avoid feedback loops in collecting ratings for the recommendation system. Apart from matrix factorization-based models for recommendation systems, Velox provides simple APIs for usage and building of ML models that enable it to incorporate other models such as SVMs and neural networks.

5. SYSTEMS FOR FEATURE ENGINEERING

Designing and choosing the right *features* for ML techniques is widely regarded as the most time-consuming and labor-intensive, yet crucial, phase

of an applied machine learning project. [8, 13, 22, 24]. While we are not aware of a formal definition, the term "feature engineering" is an umbrella term that generally encompasses all activities that transform raw data into a clean feature vector form that is needed for the implementation of an ML technique [8, 13, 18, 19]. As can be expected, feature engineering is extremely diverse, with the following activities giving a snapshot of what analysts do to engineer features: converting a text-based e-mail into a bag-of-words representation by counting word fequencies for spam detection, extracting signals from an image for face recognition, selecting a subset of features about insurance customers for churn prevention, ranking features about gene expression to detect diseases, slicing and dicing data about products and users for a recommendation system, and clustering data about account logins learn shapes in the data that can be used as features to detect malicious accounts. Typically, feature engineering is not a one-shot linear process, but rather a complex, exploratory, and cyclical process of designing, testing, and refining features [8]. Often, this involves training, inference, and testing of different ML models in the "inner loop".

Different subsets of the activities involved in feature engineering have received attention from the ML and other communities. For example, selecting a subset of features from a structured dataset to optimize some criterion is a problem known as "subset selection", which along with feature ranking is considered to be part of the larger and well-studied problem of "feature selection" [18, 19, 24]. Extracting features from unstructured text has received a lot of attention from the natural language processing community [8, 33]. However, by and large, extracting features from real-world data is considered a challenging "black art" by the ML community [8,28]. This is primarily because the decision of what constitutes "useful", or even "usable" features for an applied ML task is influenced by a complex combination of factors that span from technical to logistical, e.g., the characteristics of the data and the application, desired ML quality, ML model interpretability, constraints on time or computational resources, policies of the organization, and legal restrictions [13,22,24]. Thus, it is perhaps not surprising that analysts often end up spending a bulk of their time on feature engineering. But what might be surprising is that not much research attention has been paid to building tools that could help analysts meet the challenge of feature engineering.

5.1 Columbus

The Columbus system aims to support an analyst with the challenge of feature engineering [24, 34]. While statistical software packages and data mining toolkits offer libraries of many feature selection algorithms, Columbus recognizes that feature selection is typically not just a single algorithm, but rather an analyst-in-the-loop process that involves feature selection algorithms, descriptive statistics, and data manipulations. Thus, Columbus provides a declarative framework of operations for feature selection as a domain-specific language that can be used as a library in R itself. Feature selection programs are then translated to R, and possibly to an underlying R-based analytics systems. However, a key observation in Columbus is that there is often a lot of scope for exploiting opportunities to share computations and materialize intermediate results across the operations in a feature selection program. Thus, Columbus includes an optimizer that improves overall performance by applying a suite of optimization techniques from the classical database literature as well numerical analysis and statistical techniques. By raising the process of feature selection to a declarative level rather than procedural R scripts, and by increasing the velocity of the exploratory process, Columbus aims to make feature selection both faster and easier for analysts. Furthermore, since the "higher-level logic" of the process is available to the system, it can help manage provenance to aid in debugging.

5.2 DeepDive

The DeepDive project aims to build an end-to-end system for knowledge base construction by employing scalable learning and inference over factor graphs [33]. A key goal of the system is to make it easier for an analyst to design features that help improve the knowledge base's accuracy. Since DeepDive-based applications deal a lot with textual and other forms of unstructured data (in addition to structured data sources), DeepDive includes heavy machinery for feature engineering, especially over text. Loosely speaking, a feature for a factor graph model is a form of structured dependency between groups of random variables (also termed a "correlation" in their paper). Thus, DeepDive uses an MLN-based language that enables analysts to specify features using first-order logic statements or SQL queries. Additionally, DeepDive also handles arbitrary user-defined functions (written in a scripting language like Python) that contain code to synthesize fea-

tures. These UDFs are integrated into DeepDive's feature engineering workflows, which enables analysts to explore different features in an end-to-end fashion. DeepDive's ideas for handling feature engineering workflows were originally described in the Brainwash system vision [8]. DeepDive also provides infrastructure for inspecting and debugging the effects of different features on overall quality.

6. SYSTEMS FOR ALGORITHM SELECTION

Algorithm selection is the task of determining which specific ML technique (algorithm) is to be used for a particular application of ML. As an example, consider an analyst at an insurance company that wants to determine if a customer is going to leave the company or not – a standard problem known as *customer churn* [34]. She wants to build a *binary classification* model, but has to choose from literally dozens of ML techniques designed for this problem. Typically, analysts pick a time-tested and popular ML technique such as logistic regression, Naive Bayes, or a decision tree. Furthermore, an ML technique might have various implementations, e.g., a logistic regression model can be trained using either SGD, or a batch gradient method such as L-BFGS, or conjugate gradient [32]. Selecting an algorithm is challenging partly because the decision is usually affected by a complex combination of factors – both technical and logistical – such as time constraints, available computational resources, desired ML quality, ML model interpretability, data characteristics, how much effort the analyst is willing to invest, what toolkits the analyst can use, application-specific best practices, etc. This peculiar complexity of algorithm selection is perhaps why there is not much research on building systems that help an analyst with this challenge.

6.1 MLBase

The MLBase system aims to support an analyst with the challenge of algorithm selection [25]. Essentially, MLBase provides a declarative dataflow-based language for an analyst to specify high-level tasks, e.g., classification. MLBase then executes a bunch of internally-defined algorithms for that high-level task, and performs cross-validation to pick the best algorithm. In their paper, the authors mention that MLBase might execute different SVM models as well as an AdaBoost model for classification [25]. The system returns a summary of the execution to help an analyst understand *what* exactly was computed by MLBase. However, it is not clear why they restrict themselves to SVM models and AdaBoost. Moreover, an analyst has no way of specifying, or restricting the search space – it is hardwired arbitrarily by the system. Thus, while MLBase is one attempt to help tackle the challenge of algorithm selection, it is clear that there are many open questions and possibilities in this space.

6.2 AzureML

AzureML aims to make it easier for analysts to construct ML workflows, and is accessible over the cloud [2]. One of its functionalities is the ability to specify the training of many ML models in bulk, which is a simple way to deal with algorithm selection. The system then exploits massive parallelism to train all these models. However, it is not clear if it does anything more sophisticated other than just train many ML models. Thus, compared to MLBase, this can be viewed as lying closer to the opposite end of the specturm with respect to automating algorithm selection, wherein the analyst decides a priori what all ML models need to be trained.

7. SYSTEMS FOR PARAMETER TUNING

Most ML techniques have a number of parameters (also called *hyperparameters* in the literature [28]) that need to be specified prior to learning. For example, the coefficient vector in a logistic regression model is typically constrained using its L1-norm or L2-norm [20]. The parameter that specifies the tightness of the constraint is called the *regularization* parameter. Furthermore, SGD, which can be used to train a logistic regression model, has additional parameters such as the *stepsize*, and *decay* [14, 32]. Tuning such parameters is challenging in part because the optimization problems involved in picking the values of such parameters are usually not convex [9, 20, 32]. Thus, in practice, analysts often perform ad-hoc tuning by manually picking a set of parameter values to try, or by using "best-practice" heuristics such as a *grid search* with pre-defined intervals for each parameter [20]. However, choosing the split points for the intervals is still mostly heuristic, and the problem gets worse as the number of parameters to tune increases.

7.1 Statistical software packages

Ease of parameter tuning is one of the appeals of statistical software packages such as R and SAS. For many ML techniques, analysts can specificy the pa-

rameter tuning strategy, or easily implement their own ad-hoc strategy. Furthermore, the implementations of many ML techniques in such packages provide automatic parameter tuning, which obviates the need for an analyst to provide them. The stock off-the-shelf strategy for parameter tuning used by the implementation is a grid search with an algorithm-specific choice of intervals for parameters, followed by a validation of model quality, typically using 10-fold cross validation. Note that it is still straightforward for an analyst to implement such parameter tuning strategies themselves as an "outer loop" around the ML implementations. Coupling them with ML training simply improves the usability of the system.

7.2 MLBase

Along with algorithm selection, MLBase also aims to tackle parameter tuning. Basically, MLBase implements the same off-the-shelf parameter tuning strategies for the pre-defined set of algorithms in its search space, e.g., grid search with 10-fold cross-validation [25]. To help a user understand what exactly it computed, MLBase returns a summary. Since MLBase decouples the logical specifications from the physical execution, it can theoretically exploit opportunities to "optimize" the execution. However, it is not clear what these optimization opportunities are that are related to parameter tuning, and what MLBase implements. Furthermore, MLBase does not provide a way for an analyst to customize the search process, which statistical software packages can do. Thus, as with algorithm selection, it is clear that there are many open questions and possibilities in this space.

7.3 AzureML

AzureML also provides options that help automate parameter tuning for many ML algorithms [2]. And here too, it exploits massive parallelism to speed up the learning, in a manner similar to [15]. Analysts can specify various "parameter sweeps" which specify some constraints on the search space of the parameter values to explore. However, while it provides more customization for parameter tuning, it is not clear if it exploits any optimization opportunities to avoid redundant computations. Nevertheless, the concept of customizable parameter tuning options offers a sweet-spot between flexibility and automaticity, and we think it is worth adopting.

8. REFERENCES

[1] Apache Mahout. `mahout.apache.org`.
[2] Microsoft Azure ML. `studio.azureml.net`.
[3] Oracle R Enterprise. `www.oracle.com`.
[4] Pedro Domingos et al. `alchemy.cs.washington.edu`.
[5] Project R. `r-project.org`.
[6] SAS Report on Analytics. `sas.com/reg/wp/corp/23876`.
[7] M. Akdere et al. The Case for Predictive Database Systems: Opportunities and Challenges. In *CIDR*, 2011.
[8] M. Anderson et al. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.
[9] D. P. Bertsekas. Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey. Technical report, LIDS, MIT, 2010.
[10] Z. Cai et al. Simulation of Database-valued Markov Chains Using SimSQL. In *SIGMOD*, 2013.
[11] D. Crankshaw et al. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *CIDR*, 2015.
[12] J. Dean et al. Large Scale Distributed Deep Networks. In *NIPS*, 2012.
[13] P. Domingos. A Few Useful Things to Know about Machine Learning. *CACM*, 2012.
[14] X. Feng et al. Towards a Unified Architecture for in-RDBMS Analytics. In *SIGMOD*, 2012.
[15] Y. Ganjisaffar et al. Distributed Tuning of Machine Learning Algorithms Using MapReduce Clusters. In *LDMTA*, 2011.
[16] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning)*. The MIT Press, 2007.
[17] A. Ghoting et al. SystemML: Declarative Machine Learning on MapReduce. In *ICDE*, 2011.
[18] I. Guyon et al. *Feature Extraction: Foundations and Applications*. New York: Springer-Verlag, 2001.
[19] I. Guyon et al. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, Mar. 2003.
[20] T. Hastie et al. *Elements of Statistical Learning: Data mining, inference, and prediction*. Springer-Verlag, 2001.
[21] J. Hellerstein et al. The MADlib Analytics Library or MAD Skills, the SQL. In *VLDB*, 2012.
[22] S. Kandel et al. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE TVCG*, 2012.
[23] M. L. Koc and C. Ré. Incrementally Maintaining Classification Using an RDBMS. In *VLDB*, 2011.
[24] P. Konda et al. Feature Selection in Enterprise Analytics: A Demonstration using an R-based Data Analytics System. In *VLDB*, 2013.
[25] T. Kraska et al. MLbase: A Distributed Machine-learning System. In *CIDR*, 2013.
[26] A. Kumar et al. Model Selection Management Systems: The Next Frontier of Advanced Analytics. *ACM SIGMOD Record*, December 2015.
[27] Y. Low et al. GraphLab: A New Framework For Parallel Machine Learning. In *UAI*, 2010.
[28] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
[29] M. Nikolic et al. LINVIEW: Incremental View Maintenance for Complex Analytical Queries. In *SIGMOD*, 2014.
[30] F. Niu et al. Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS. In *VLDB*, 2011.
[31] F. Niu, B. Recht, C. Ré, and S. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.
[32] J. Nocedal et al. *Numerical Optimization*. Springer, 2006.
[33] C. Ré et al. Feature Engineering for Knowledge Base Construction. *IEEE Data Engineering Bulletin*, 2014.
[34] C. Zhang et al. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*, 2014.
[35] C. Zhang and C. Ré. Towards High-throughput Gibbs Sampling at Scale: A Study Across Storage Managers. In *SIGMOD*, 2013.
[36] Y. Zhang et al. I/O-Efficient Statistical Computing with RIOT. In *ICDE*, 2010.
[37] M. Zinkevich et al. Parallelized Stochastic Gradient Descent. In *NIPS*, 2010.