Landset *et al. Journal of Big Data (2015) 2:24*
DOI 10.1186/s40537-015-0032-1

Journal of Big Data
a SpringerOpen Journal

SURVEY PAPER Open Access

A survey of open source tools for machine learning with big data in the Hadoop ecosystem

Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter* and Tawfiq Hasanin

*Correspondence:
arichter@fau.edu
Florida Atlantic University,
777 Glades Road, Boca Raton,
FL 33431, USA

Abstract

With an ever-increasing amount of options, the task of selecting machine learning tools for big data can be difficult. The available tools have advantages and drawbacks, and many have overlapping uses. The world's data is growing rapidly, and traditional tools for machine learning are becoming insufficient as we move towards distributed and real-time processing. This paper is intended to aid the researcher or professional who understands machine learning but is inexperienced with big data. In order to evaluate tools, one should have a thorough understanding of what to look for. To that end, this paper provides a list of criteria for making selections along with an analysis of the advantages and drawbacks of each. We do this by starting from the beginning, and looking at what exactly the term "big data" means. From there, we go on to the Hadoop ecosystem for a look at many of the projects that are part of a typical machine learning architecture and an understanding of how everything might fit together. We discuss the advantages and disadvantages of three different processing paradigms along with a comparison of engines that implement them, including MapReduce, Spark, Flink, Storm, and H_2O. We then look at machine learning libraries and frameworks including Mahout, MLlib, SAMOA, and evaluate them based on criteria such as scalability, ease of use, and extensibility. There is no single toolkit that truly embodies a one-size-fits-all solution, so this paper aims to help make decisions smoother by providing as much information as possible and quantifying what the tradeoffs will be. Additionally, throughout this paper, we review recent research in the field using these tools and talk about possible future directions for toolkit-based learning.

Keywords: Machine learning, Big data, Hadoop, Mahout, MLlib, SAMOA, H2O, Spark, Flink, Storm

Background

As the price of data storage has gone down and high performance computers have become more widely accessible, we have seen an expansion of machine learning (ML) into a host of industries including finance, law enforcement, entertainment, commerce, and healthcare. As theoretical research is leveraged into practical tasks, machine learning tools are increasingly seen as not just useful, but integral to many business operations.

The goal of machine learning is to enable a system to learn from the past or present and use that knowledge to make predictions or decisions regarding unknown future events. In the most general terms, the workflow for a supervised machine learning task consists of three phases: build the model, evaluate and tune the model, and then put the model into production. An example of this workflow is in Fig. 1.

At the heart of machine learning is the data that powers the models, and the new era of Big Data is catapulting machine learning to the forefront of research and industry applications. The meaning of the term "big data" is still the subject of some disagreement, but it generally refers to data that is too big or too complex to process on a single machine. We live in an age where data is growing orders of magnitude faster than ever before. According to International Data Corporation's annual Digital Universe study [1], the amount of data on our planet is set to reach 44 zettabytes (4.4×1022 bytes) by 2020 which would be ten times larger than it was in 2013. While no single entity is working with data at this magnitude, many industries are still generating data too large to be processed efficiently using traditional techniques. Ancestry.com, for example, stores billions of records totaling about 10 petabytes of data [2]. With such a growth rate in data production, the challenge faced by the machine learning community is how to best efficiently process and learn from big data. Popular machine learning toolkits such as R [3] or Weka [4] were not built for these kinds of workloads. Although Weka has distributed implementations of some algorithms available, it is not on the same level as tools that were initially designed and built for terabyte-scale. Hadoop [5], a popular framework for working with big data, helps to solve this scalability problem by offering distributed storage and processing solutions. While Hadoop is just a framework for processing data, it provides a very extensible platform that allows for many machine learning projects and applications; the focus of this paper is to present those tools.

The proliferation of big data has forced us to rethink not just data processing frameworks, but implementations of machine learning algorithms as well. Choosing the appropriate tools for a particular task or environment can be daunting for two reasons. First, the increasing complexity of machine learning project requirements as well as of the data itself may require different types of solutions. Second, often developers will find the selection of tools available to be unsatisfactory, but instead of contributing to existing open source projects, they begin one of their own. This has led to a great deal of fragmentation among existing big data platforms. Both of these issues can contribute to the difficulty of building a learning environment, as many options have overlapping use

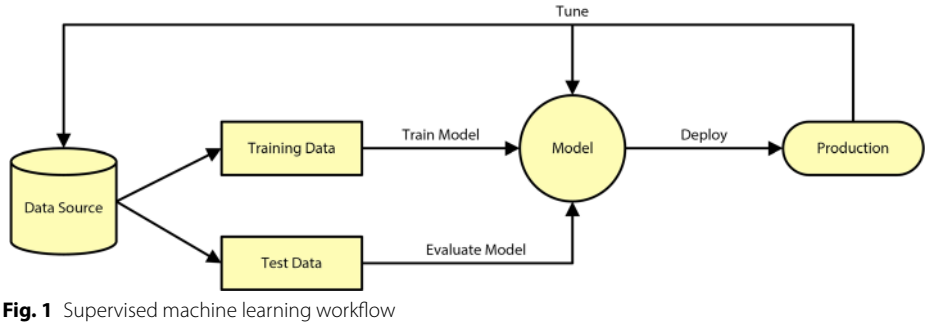

Fig. 1 Supervised machine learning workflow

cases, but diverge in important areas. Because there is no single tool or framework that covers all or even the majority of common tasks, one must consider the trade-offs that exist between usability, performance, and algorithm selection when examining different solutions. There is a lack of comprehensive research on many of them, despite being widely employed on an enterprise level and there is no current industry standard.

The goal of this paper is to facilitate these decisions by providing a comprehensive review of the current state-of-the-art in open source scalable tools for machine learning. Recommendations are offered for criteria with which to evaluate the various options, and comparisons are provided between various open source data processing engines as well as ML libraries and frameworks. This paper presumes that the reader has a basic knowledge of machine learning concepts and workflows. It is intended for people who have experience with machine learning and want information on the different tools available for learning from big data. The paper will be useful to anyone interested in big data and machine learning, whether a researcher, engineer, scientist, or software product manager.

The remainder of this paper will be organized as follows: The section titled "Understanding big data" provides background on the problems that may arise when working with big data, and the "Hadoop ecosystem" section serves as an explanation and overview of the Hadoop ecosystem with a focus on tools that can help solve big data problems. The "Data processing engines" section examines different data processing paradigms and outlines criteria for evaluation. "Machine learning toolkits" discusses criteria for evaluation of machine learning tools and libraries, and "Evaluation of machine learning tools" provides an in-depth analysis of specific frameworks that can be used with the processing platforms. The "Suggestions for future work" section contains a discussion of key elements missing among the major toolkits and the final section presents conclusions from this survey.

Understanding big data

The term "big data" has become a buzzword and as such, it is often overused and misunderstood. While the frameworks we discuss in this paper are able to effectively process data of varying sizes and complexities, they were designed with very large data in mind and may not be the best choice for certain smaller projects. For this reason, the first step in choosing between big data frameworks is to determine if they are needed. In order to do this, it is important to have an understanding of what constitutes big data. This section provides definitions of big data and discusses the challenges associated with it.

There is no universally agreed-upon definition of big data, but the more widely accepted explanations tend to describe it in terms of the challenges it presents. This is sometimes referred to as the "big data problem." In 2001, Laney [6] described three-dimensions of data management challenges. This characterization, which addresses volume, velocity, and variety, is frequently documented in scientific literature. These three dimensions (commonly referred to as the 3 V's) can be understood as follows:

- *Volume* is the most obvious of the three, referring to the size of the data. The massive volumes of data that we are currently dealing with has required scientists to rethink

storage and processing paradigms in order to develop the tools needed to properly analyze it.

- *Velocity* addresses the speed at which data can be received as well as analyzed. In the "Data processing engines" section, we discuss the differences between batch processing, which works on historical data, and stream processing, which analyzes the data in real-time as it is generated. This also refers to the rate of change of data, which is especially relevant in the area of stream processing.
- *Variety* refers to the issue of disparate and incompatible data formats. Data can come in from many different sources and take on many different forms, and just preparing it for analysis takes a significant amount of time and effort.

In the years since Laney's paper was published, numerous people have proposed additions to this list and many refer to four or five V's, adding in Value or Veracity [7]. However, we are skeptical that these additions add to an overall understanding of big data, so we focus our discussion here to the original three.

In 1997, Cox and Ellsworth [8] were among the first authors in scientific literature to discuss big data in the context of modern computing. Their work focused on data visualization, but their observations about the big data problem can easily be extrapolated to general data analytics and machine learning. The big data problem, according to them, consists of two distinct issues:

- *Big data collections* are aggregates of multiple datasets that are individually manageable, but as a group are too large to fit on disk. The datasets in these collections typically come from different sources, are in disparate formats, and are stored in separate physical sites and in different types of repositories.
- *Big data objects* are individual datasets that by themselves are too large to be processed by standard algorithms on available hardware. Unlike collections, they typically come from a single source.

Today, the problem of big data collections is often solved through distributed storage systems, which are designed to carefully control access and management in a fault-tolerant manner. One solution for the problem of big data objects in machine learning is through parallelization of algorithms. This is typically accomplished in one of two ways [9]: data parallelism, in which the data is divided into more manageable pieces and each subset is computed simultaneously, or task parallelism, in which the algorithm is divided into steps that can be performed concurrently.

It is not uncommon to encounter big collections of big objects as data grows and becomes more widely available. This, coupled with unprecedented access to computing power through more affordable high performance machines as well as cloud services, is opening up many new opportunities for machine learning research. Many of these new directions utilize increasingly complex workflows which require systems built using a combination of state-of-the art tools and techniques. One option for such a system is to use projects from the Hadoop Ecosystem. The remainder of this paper provides detailed information about these projects and discusses how they can be utilized together to build an architecture capable of efficiently learning from data of this magnitude.

Hadoop ecosystem

Many people consider the terms *Hadoop* and *MapReduce* to be interchangeable, but this is not entirely accurate. Hadoop was initially introduced in 2007 as an open source implementation of the MapReduce processing engine linked with a distributed file system [10], but it has since evolved into a vast web of projects related to every step of a big data workflow, including data collection, storage, processing, and much more. The amount of projects that have been developed to either complement or replace these original elements has made the current definition of Hadoop unclear. For this reason, we often hear reference to the *Hadoop Ecosystem* instead, which encompasses these related projects and products. To fully understand Hadoop, one must look at both the project itself and the ecosystem that surrounds it. The Hadoop project itself currently consists of four modules [10]:

- *Hadoop distributed file system (HDFS)* A file system designed to store large amounts of data across multiple nodes of commodity hardware. HDFS has a master–slave architecture made up of data nodes which each store blocks of the data, retrieve data on demand, and report back to the name node with inventory. The name node keeps records of this inventory (references to file locations and metadata) and directs traffic to the data nodes upon client requests. This system has built-in fault tolerance, typically keeping three or more copies of each data block in case of disk failure. Additionally, there are controls in case of name node failure as well, in which a system will either have a secondary name node, or will write backups of metadata to multiple file systems.
- *MapReduce* Data processing engine. A MapReduce job consists of two parts, a map phase, which takes raw data and organizes it into key/value pairs, and a reduce phase which processes data in parallel. A detailed discussion of this processing approach can be found in the following section.
- *YARN ("Yet Another Resource Negotiator")* [11] Prior the addition of YARN to the Hadoop project in version 2.0, Hadoop and MapReduce were tightly coupled, with MapReduce responsible for both cluster resource management and data processing. YARN has now taken over the resource management duties, allowing a separation between that infrastructure and the programming model. With YARN, if an application wants to run, its client has to request the launch of an application manager process from the resource manager, which then finds a node manager. The node manager then launches a container which executes the application process. For any readers who are familiar with previous versions of Hadoop, the jobtracker responsibilities from MapReduce are now YARN, split between the resource manager, application master, and timeline server (which stores application history), while the old tasktracker responsibilities are handled by the node managers. This change has improved upon many of the deficiencies present in the old MapReduce. YARN is able to run on larger clusters, more than doubling the amount of jobs and tasks it can handle before running into bottlenecks [10]. Finally, YARN allows for a more generalized Hadoop which makes MapReduce just one type of YARN application. This means it can be left out altogether in favor of a different processing engine.

- *Common* [12] A set of common utilities needed by the other Hadoop modules. It has native shared libraries that include Java implementations for compression codecs, I/O utilities, and error detection. Also included are interfaces and tools for configuration of rack awareness, authorization of proxy users, authentication, service-level authorization, data confidentiality, and the Hadoop Key Management Server (KMS).

The Hadoop ecosystem is made up of a vast array of projects built on top of and around the core modules described above. These projects have been designed to aid researchers and practitioners in all aspects of a typical data analysis or machine learning workflow. Several companies such as Cloudera [13], Hortonworks [14], and MapR [15] offer distributions of Hadoop which bundle a number of these projects. Free and enterprise versions of the software bundles are available.

The general structure of the ecosystem can be described in terms of three layers: storage, processing, and management. While the primary focus of this paper is on tools that reside in the processing layer, it is important to understand the context of how they can be used in a workflow by looking at the makeup of the ecosystem as a whole. An example of how tools for different tasks may fit together as part of an analytical stack is shown in Fig. 2. The specific projects listed inside this diagram and discussed in this section are examples of commonly used tools, but since the ecosystem is made up of well over 100 projects, it is not meant to be a comprehensive list. Readers who wish to learn more about the tools not discussed in this paper are encouraged to refer to the Hadoop website or [10] for more information.

Storage layer

The storage layer resides at the lowest level of this stack, and by default it includes the HDFS described previously. There are also a variety of other options for distributed data storage which either run on top of the HDFS or work as standalone systems. The HDFS is not a database, but a file storage system designed for a specific purpose, and it doesn't include all of the functionality that some of the other data storage solutions have. HDFS is known for its scalability and fault tolerance, and is a good option for historical data that does not need to be edited or accessed frequently, but there are several limitations that may impact Hadoop users, in particular one for whom fast random reads or writes

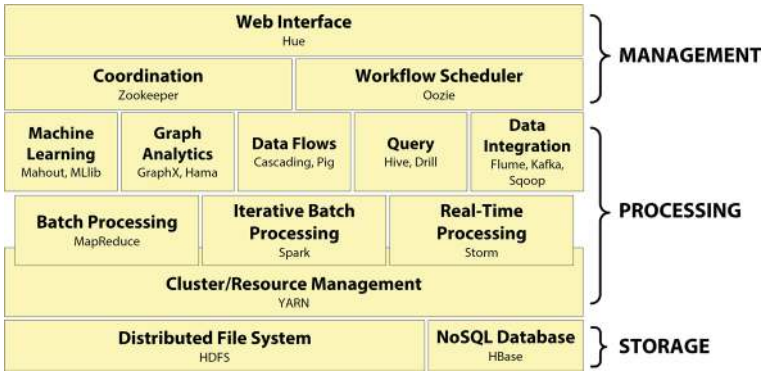

Fig. 2 The Hadoop ecosystem

are a priority. Tools do exist to support SQL queries, and they will be discussed in the next subsection. HDFS operates on a write-once, read-many paradigm, so if changes are needed on even a single data point, the entire file must be rewritten. For these reasons, many choose to add one or more storage solutions to their architecture.

Non-relational databases, collectively referred to as NoSQL (Not only SQL), can be suitable for machine learning tasks, because they support nested, semi-structured, and unstructured data. Databases in this category typically use one of four basic types of data models and the choice of database will ultimately depend on the data being stored as well as the demands of the project for which it is being used. The four types of databases include:

1. *Key-value stores* This is the simplest of the four models, implemented as what is essentially a large hash table. Each data item has a unique key pointing to it. They are fast and highly scalable. Some examples of databases built on this model include Voldemort [16] or Redis [17].
2. *Document stores* These can be thought of as nested key-value stores, where a key points to a collection of key-value stores rather than simply a value. Examples include CouchDB [18] and MongoDB [19].
3. *Column-oriented* Data is stored in columns rather than the typical row/column structure. Columns are grouped into column families. HBase [20] and Cassandra [21] are both examples of column-oriented data stores.
4. *Graph-based models* Designed for data that can be represented as a graph and can be used for tasks such as network analysis. They are more flexible than the other models, with no tables or rows. Examples include Titan [22], Neo4J [23], and OrientDB [24].

Processing layer

The processing layer is where the actual analysis takes place. The foundation of this layer is YARN, which allows one or more processing engines to run on a Hadoop cluster. Processing engines will be discussed in detail in the next section. In addition to the processing engines, this layer includes a number of different tools that can be used for machine learning and data analysis. ML libraries and frameworks will be discussed in the "Machine learning toolkits" and "Evaluation of machine learning tools" sections.

In addition to processing frameworks and libraries, this layer includes tools for data movement and interaction. Examples of this are data integration tools such as Flume [25], Kafka [26], and Sqoop [27]. Flume handles collection, aggregation, and movement of log data into HDFS. Kafka is a distributed publish-subscribe messaging system on top of HDFS, and Sqoop transfers bulk data between the HDFS and relational databases. On the interaction side, we find query engines such as Hive [28] and Drill [29]. Hive queries data stored in the HDFS and NoSQL databases using HiveQL, an extension of ANSI SQL which is similar to MySQL. Metadata for tables and partitions is kept in the Hive Metastore. Drill performs queries using ANSI SQL and supports self-describing data, in which schema is discovered dynamically on read, eliminating the need for data transformation which is a time-consuming process. It also offers plugins for configuration with

Hive allowing for use of the metastore and any user-defined functions that were previously built.

One of the main drawbacks to using MapReduce is that many algorithms do not translate easily into this pattern [30]. Cascading [31] and Pig [32] aim to address this by offering high level abstractions which hide some of the complexity inherent to MapReduce jobs, thereby simplifying the programming process. Pig offers an execution framework and dataflow language called Pig Latin, a scripting language. It supports user-defined functions written in Python, Java, JavaScript, and Ruby which are then translated to MapReduce jobs. In addition to not requiring the user to think in terms of map and reduce, it offers a multi-query execution to batch statements, significantly cutting down on the amount of code the programmer has to write. It can be used for machine learning tasks, most notably used by Twitter, whose engineers note that it allows learning tasks to be executed and tuned using only a few lines of code [33]. Pig runs on MapReduce and Tez [34], which is an abstraction of MapReduce that represents data flow in the form of a directed acyclic graph.

A similar project, Cascading, offers an Application Programming Interface (API) that abstracts the traditional keys and values into tuples with field names and offers a number of operations on the tuples that help developers build complex applications more easily and in less time. They have also announced that upcoming releases will offer support for Spark, Storm, and Tez. Cascading primarily supports programming in Java, but also offers APIs for ANSI SQL, Predictive Model Markup Language (PMML), Scala, Clojure, JRuby, and Python. It also supports easy integration of a large number of different data sources.

Management layer

The management layer includes tools for user interaction and high-level organization. These include scheduling, monitoring, coordination, and user interface. Oozie [35], a workflow scheduler, manages jobs for many of the tools in the processing layer, including processing engines, Pig, Sqoop, and Hive, among others. For complex workflows which require multiple jobs and tools, it specifies a sequence of actions and coordinates between them to complete the tasks. It also facilitates scheduling of jobs which need to run on regular intervals.

Zookeeper [36] is a service for coordination and synchronization of distributed systems. It provides tools to handle coordination of data and protocols and is able to handle partial network failures, which are commonplace in distributed systems. It includes APIs for Java and C, and also has bindings for Perl, Python, and REST clients.

Hue [37], a web interface for Hadoop projects, supports many of the more widely used components of the Hadoop ecosystem. It features file browsers for HDFS and HBase and a job browser for MapReduce/YARN. It can be used to manage interactions with Hive, Pig, Sqoop, Zookeeper, and Oozie, and in addition also offers tools for data visualization. It is compatible with any version of Hadoop and is available in all of the major Hadoop distributions.

Machine learning without Hadoop

While Hadoop is ubiquitous as a big data framework, there are a number of other open source options for machine learning that do not use it at all. MOA (Massive Online Analysis) [38] is a project related to Weka, which offers online stream analysis on a number of Weka algorithms and with the same user interface.[1] MADlib is a collection of SQL-based algorithms designed to run at scale within the database rather than porting data between multiple runtime environments. It includes clustering, classification, regression, and topic models as well as tools for validation [39]. Dato, formerly GraphLab, is a standalone product that can be connected with Hadoop for graph analysis and ML tasks. It was fully open source, but in late 2014, they transitioned into a commercial product. Their C++ processing engine Dato Core [40] has been released to the community on Github along with their interprocess communication library (for translating between C++ and Python) and graph analytics implementations. Their machine learning libraries are unavailable outside of their enterprise packages. Distributed processing on Hadoop enables large-scale learning, and the goal of this paper is to profile tools that can do exactly that. Non-distributed tools for machine learning are widely available, and are thus more mature for use in projects that do not handle Big Data. Using Hadoop for smaller scale workloads would not be advised, as there is overhead to distributed processing, and there are fewer algorithm and implementation choices. This paper aims to profile tools that can effectively handle Big Data, therefore projects that do not run on Hadoop are outside the scope of this paper and will not be discussed in further detail.

Data processing engines

When MapReduce was introduced in 2004 by Google engineers [41], it had some early critics [42], but was considered by many to be revolutionary. Regardless of the differing opinions on the value of this idea, it paved the road for Hadoop, which has played a significant role in ushering in the big data era. In more recent years, MapReduce has begun to fall out of favor, particularly in the machine learning community, due to its high overhead costs, lack of speed, and the fact that many machine learning tasks do not easily fit into the MapReduce paradigm. In 2014, Google announced that it was being phased out in favor of other projects [43]. Since MapReduce has been decoupled from Hadoop through YARN, it is now a lot easier to work with a new engine on an existing cluster, and over the course of the past few years, a number of projects have been introduced that attempt to solve the issues inherent in MapReduce.

The processing models used for many of these may be categorized as either batch or streaming. A third model, known as bulk-synchronous parallel (BSP), is used for iterative graphing tasks, but will not be discussed in detail in this paper. While graph algorithms are related to ML, they are used more for traditional analytics and the focus of this paper is on other types of learning tasks. Examples of tools which employ the BSP model are Apache Giraph [44] and Apache Hama [45]. It should be noted though that Giraph has not had a commit since mid-2013. Both projects are open source implementations of Google's Pregel [46].

[1] A related project, SAMOA, does use Hadoop and is discussed in this paper.

The remainder of this section will discuss some of the more widely used projects which leverage the batch and streaming paradigms. A high-level overview of these projects is in Table 1. In addition to the underlying processing approach used, here are several important considerations for evaluation of these tools:

1. *Latency* This refers to the amount of time between starting a job and getting initial results. Speed may not be important for every project. If a project is not time-sensitive, a batch system may be preferred for its simplicity, but for projects that are require real time or near-real time results, a streaming platform would be advised.
2. *Throughput* Throughput measures the amount of work done over a given time period. This can be thought of as a measure of efficiency.
3. *Fault tolerance* All of the platforms discussed in this paper are fault-tolerant but the methods which they use to achieve that may vary. We look at the mechanisms that are in place to detect failures, as well as how the platform is able to recover after such a failure occurs.
4. *Usability* Despite the interfaces and abstractions discussed in the previous section and libraries for machine learning that will be discussed later in this paper, the typical user will spend a good deal of time interacting with the engine itself. With this in mind we ask, how difficult is it to install and configure? What interface language(s) does it use? How difficult is it to program for?
5. *Resource expense* In this paper, we consider expense mostly in terms of time involved from setting up a cluster to deploying the model and maintaining it after the fact. Most of this is covered in usability. While we don't examine financial costs in this paper, they are important to consider as well. This will depend on whether the user has access to a high performance computing cluster. Purchasing the necessary equipment is not trivial, and the resources needed by processors can vary, so one decision may affect the other. Alternatively, clusters can be set up on cloud services such as Amazon EC2 [47] or Microsoft Azure [48], which charge on-demand prices based on compute time and storage space used.

Table 1 Data processing engines for Hadoop

	Current stable release (as of June 1, 2015)	Execution model	Supported languages	Associated ML tools	In-memory processing	Low latency	Fault tolerance	Enterprise support
MapReduce	2.7.0	Batch	Java	Mahout	×	×	✓	×
Spark	1.3.1	Batch, streaming	Java, Python, R, Scala	MLlib, Mahout, H_2O	✓	✓	✓	✓
Flink	0.8.1	Batch, streaming	Java, Scala	Flink-ML, SAMOA	✓	✓	✓	×
Storm	0.9.4	Streaming	Any	SAMOA	✓	✓	✓	×
H_2O	3.0.0.12	Batch	Java, Python, R, Scala	H_2O, Mahout, MLlib	✓	✓	✓	✓

6. *Scalability* All of the processing engines discussed in this paper were designed to be scalable, but the different methods employed have varying degrees of success. For this reason, it is important to examine whether there are bottlenecks when data input or cluster sizes grow. Additionally, these engines were designed for very large data, but many real-world use cases involve at least some processing of smaller datasets. We look at how these are handled as well.

The approaches to processing differ in terms of throughput and resource expense, and there are additional platform-dependent features that should also be used for evaluation of these projects. To provide a comprehensive comparison, fault-tolerance methods, scalability, efficiency, interface language, and usability are covered below.

MapReduce

The MapReduce approach to machine learning performs batch learning, in which the training data set is read in its entirety to build a learning model. The biggest drawback to this batch model is a lack of efficiency in terms of speed and computational resources. In a typical batch-oriented workflow, the set of training data is read from the HDFS to the mapper as a set of key-value pairs. The output, a list of keys and their associated values, is written to disk. In a classification task, for example, the initial key-value pair might be a filename and a list of instances, and the intermediate output from the mapper would be a list of each instance with its associated class. This intermediate data is then read into one or more reducers to train a model based on this list. The final model is then once again written to disk. This process is illustrated in Fig. 3a.

These frequent I/O operations can become very expensive in terms of time, computational resources, and network bandwidth. Any model parameters that need to be tuned after the initial evaluation stage further add to the costs. These issues become more apparent in cases where it is necessary to update models with changing data, which is often the case in real-world ML production environments. While this approach may be suitable for certain projects such as analyzing past events, it becomes problematic when data evolves, as the full process must be repeated each time a model requires updating. Data must be in its final form before beginning a MapReduce job, as the mechanism does not have the ability to wait for new data to be generated. MapReduce is compatible with the Mahout library for ML, and the programming interfaces discussed in the previous section can be used as well. We mentioned Twitter, who built their analytics stack around Pig and performs ML tasks using Pig's user-defined functions [33]. There is also a framework called Conjecture [49], which was developed by Etsy.com engineers for parallelized online learning in Scalding, a Scala wrapper for Cascading.

The fault tolerance mechanism employed by MapReduce is achieved through data replication, which can affect scalability by increasing the size of data even further. The need for data replication has been found to be responsible for 90 % of the running time of machine learning tasks in MapReduce [50] and is perhaps the biggest impediment to fast data processing. Another deficiency of MapReduce is that it does not easily allow for iterative processing, making it unsuitable for many machine learning projects. While it is possible to achieve iterative computation in MapReduce, this must be programmed manually through multiple MapReduce jobs requiring careful orchestration of execution

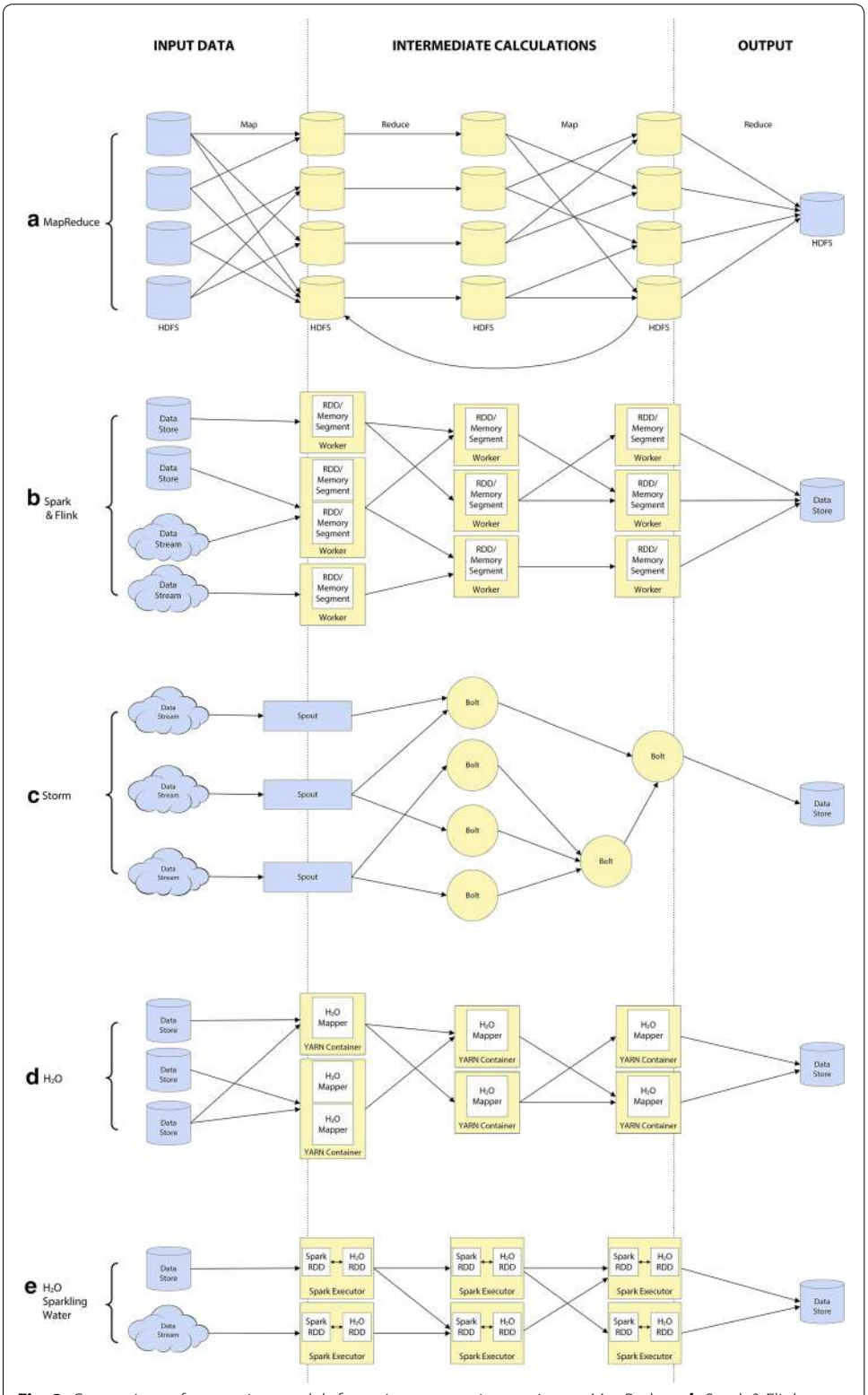

Fig. 3 Comparison of processing models for various processing engines: **a** MapReduce, **b** Spark & Flink, **c** Storm, **d** H_2O, and **e** H_2O with Sparkling Water

[51]. This process is complicated and unable to address any of the previously discussed issues relating to computational resources.

HaLoop [51], developed at the University of Washington, was an early project aimed at addressing these concerns via a programming interface which handles loop control and task scheduling. However, it lacks ongoing development, and is only compatible with older versions of Hadoop [52].

Spark

Spark [53], which was initially developed at the University of California, Berkeley [54] and is now an Apache top-level project, is based on MapReduce but addresses a number of the deficiencies described above. Like HaLoop, it supports iterative computation and it improves on speed and resource issues by utilizing in-memory computation. Spark's approach to processing has seen widespread adoption in both research and industry. The main abstractions used in this project are called Resilient Distributed Datasets (RDD), which store data in-memory and provide fault tolerance without replication [50]. RDDs can be understood as read-only distributed shared memory [55]. This model, illustrated in Fig. 3b, streamlines the learning process through in-memory caching of intermediate results, significantly cutting down on the number of read and write operations necessary.

The RDD API was extended in 2015 to include DataFrames, which allow users to group a distributed collection of data by column, similar to a table in a relational database. They can be thought of as RDDs with Schema [56]. For example, an RDD of key-value pairs can be converted into a DataFrame which is represented as a table with one column each for key and value. For users familiar with R or Python, the implementations are similar. DataFrames can be created from an existing RDD, Hive table, HDFS or a number of other data sources.

Spark's speed was demonstrated in October 2014, when it won the Daytona GraySort Benchmark Contest [57]. The previous record was held by Hadoop/MapReduce, for sorting 102.5 TB on 2100 nodes in 72 min. Spark sorted 100 TB on 206 nodes in only 23 min, three times faster with one tenth the number of machines. It was then used to sort a petabyte in 234 min on 190 nodes (though this wasn't an official part of the contest and was not posted with the winners) [58]. Additionally, it has been noted that Spark is easier to program [50, 59] and part of that reason is due to the fact that it can be coded in Java, R, Python, or Scala. For machine learning tasks, Spark ships with the MLlib [60] and GraphX [61] libraries and the latest version of the Mahout [62] library offers a number of Spark implementations as well.

In [59], Spark's performance was tested against three other machine learning platforms, SimSQL, GraphLab, and Giraph. They were run through an extensive set of tests in which they each trained five complex models on clusters of increasing size. The study compared running times on each platform for each cluster size, as well as how much code was necessary for each implementation. The results of the experiments varied, but generally showed Spark to be slower than the graphing implementations but faster than SimSQL. Though implementation was slower, it required far less code than the graphing platforms on all experiments. Additionally, for Spark, they examined the running time in both Java and Python for comparison. While they found the Python implementation to be easy to use and the code to be clean and succinct, they noted that it was significantly

slower than Java on most tests. The exception to this was one problem with 100-dimensional linear algebra, in which Java was eight times slower than Python, which was presumably an issue with Java, rather than with Spark's runtime. The authors noted that Spark required a good deal of tuning and experimentation to get large or complicated problems working, and they were unable to figure this out for all problems, causing failures on several tests. They could not agree on the reason for these problems, but one theory put forth attributed these failures to heavy reliance on techniques like lazy evaluation for speed and job scheduling. It should be noted, however, that Spark version 0.7.3 was used in these experiments and many improvements have been made to the platform (which is now in version 1.3.1) since then. Many of Spark's issues can be reasonably attributed to the fact that it is still young. There is a large team of contributors working on it all the time, so issues are often resolved even before studies are published.

Other concerns about Spark's approach deal with the distribution of data across nodes. Data transfers take place throughout the network, and because of the job isolation mechanism present, only one driver can serve requests to all of its RDDs, potentially leading to a bottleneck within the network when there are multiple requests to multiple nodes [52, 63, 64]. However, a 2015 study by Ousterhout et al. [65] used block-time analysis to identify performance bottlenecks in Spark and they found that improving network performance only had a minimal effect on job completion time while the real bottlenecks were actually occurring on the CPU rather than I/O as previously thought.

While the iterative batch approach to data processing improves on many of the deficiencies of the MapReduce paradigm, it still does not offer the ability to process data in real-time. Online data processing may be useful for projects such as clickstream analysis or event detection. Spark offers Spark Streaming, which uses micro-batching, a technique that may be thought of as a simulation of real-time processing. In this approach, an incoming stream is packaged into sequences of small chunks of data, which can then be processed by a batch system [66]. While this may be adequate for many projects, it is not a true real-time system. It is noted in [67] that this approach makes load balancing easier and is more robust to node failures. Additionally, the authors mention that while this model is slower than true streaming, the latency can be minimized enough for most real-world projects. Spark also offers integration of its streaming and batch options for more powerful interactive applications.

Storm

Storm [68] is used for processing data in real-time and was initially conceived to overcome deficiencies of other processors in collecting and analyzing social media streams [69]. Development on Storm began at BackType, a social media analytics company and continued at Twitter after a 2011 acquisition. The project was open sourced and became an Apache top-level project in September 2014 [70]. The machine learning community has been placing growing importance on real-time processing [71], and as a result, Storm is seeing increased adoption both in production and in research environments.

The Storm architecture consists of spouts and bolts. A spout is the input stream (e.g. Twitter streaming API), while bolts contain most of the computation logic, processing data in the form of tuples from either the spout or other bolts. Networks of spouts and bolts, which are represented as directed graphs, are known as topologies. An example

of this is in Fig. 3c. The project is primarily implemented in Clojure, but initially used Java for all APIs to encourage more widespread adoption. It now includes Thrift [72], a framework for cross-language development, which allows topologies to be defined and submitted using any programming language [73]. Storm uses real-time streaming, but also offers micro-batch via its Trident API.

Fault tolerance is achieved by way of the topology: Spouts will keep messages in their output queues until the bolts acknowledge them. Messages will continue to be sent out until they are acknowledged, at which time they will be dropped out of the queue. A master node, known as Nimbus because it runs the Nimbus daemon, tracks the heartbeats of worker nodes. If a worker node dies, then Nimbus will reassign the workers to another node. Nimbus also handles the responsibility of assigning tasks to workers, similar to jobtracker in MapReduce. The biggest difference is if the jobtracker dies, all running jobs are lost, but if Nimbus dies, it is automatically restarted [74].

Storm was built as a stand-alone system independent from Hadoop, but since Hadoop moved to YARN, work has been done to integrate the two projects. Hortonworks added Storm to their Hadoop distribution beginning in version 2.1 and Yahoo! is working on an integration as well [75]. The principal developer of Storm, Nathan Marz, coined the term "Lambda Architecture" in [76], describing a generalized approach to combine multiple paradigms into one system by breaking down processing into three layers: batch, serving, and speed. The batch layer stores the master dataset and computes views which are sent to the serving layer for indexing and keeping track of the most current results. The speed layer looks at new data only, as it arrives, and makes updates in real-time. New data is sent to both the batch layer and the speed layer for computation and results from each are merged when the system is queried. In terms of the processing engines we have discussed so far, the lambda architecture can be seen as a way to quickly run jobs on MapReduce and Storm simultaneously and combine the results. This unifies the processing of both real-time and historical data.

Storm does not ship with a machine learning library, but SAMOA, a platform for mining big data streams, currently has implementations for classification and clustering algorithms running on Storm. H_2O [77] has also offered a way to link the two projects [78]. Others have created their own implementations of various learning algorithms. For example, Wasson and Sales [79] describe the implementation of a particle learning algorithm built on Storm. Trident-ML [80] offers a library of learning algorithms built on Storm, but has not been updated since early 2014.

Flink

Flink [81] was developed at the Technical University of Berlin under the name Stratosphere [82]. It graduated the Apache incubation stage in January 2015 and is now a top-level project. It offers capability for both batch and stream processing, thus allowing for the implementation of a Lambda Architecture as described above. It is a scalable, in-memory option that has APIs for both Java and Scala. It has its own runtime, rather than being built on top of MapReduce. As such, it can be integrated with HDFS and YARN, or run completely independent from the Hadoop ecosystem. Flink's processing model applies transformations to parallel data collections [83, 84]. Such transformations generalize *map* and *reduce* functions, as well as functions such as join, group, and iterate.

Also included is a cost-based optimizer which automatically selects the best execution strategy for each job. Flink is also fully compatible with MapReduce, meaning it can run legacy code with no modifications [81].

Like Spark, Flink also offers iterative batch as well as streaming options, though their streaming API is based on individual events, rather than the micro-batch approach that Spark uses. This is the same model that Storm uses for true real-time processing. Connectors are offered which allow for processing data streams from Kafka, RabbitMQ (a platform-independent messaging system), Flume, Twitter, and user-defined data sources.

The project is still in its infancy but machine learning tools are in development. FlinkML [85], a machine learning library, was introduced in April 2015. Additionally, an adapter is available for the SAMOA library, which offers learning algorithms for stream processing. Ni [55] performed a comprehensive comparison of the Flink and Spark platforms and examined differences from a theoretical perspective as well as a practical one. In general, Spark was found to be superior in the areas of fault tolerance and handling of iterative algorithms, while Flink's advantages were the presence of optimization mechanisms and better integration with other projects. In terms of practical execution, Flink used more resources but was able to finish jobs in less time. Flink has undergone major changes since this study was published and more updated comparisons are needed. The Flink team published benchmark results using Grep and PageRank, and Flink's execution was significantly faster than that of Spark [86], but independent tests are needed to verify these claims.

H_2O

H_2O is an open source framework that provides a parallel processing engine, analytics, math, and machine learning libraries, along with data preprocessing and evaluation tools. Additionally, it offers a web-based user interface, making learning tasks more accessible to analysts and statisticians who may not have strong programming backgrounds. For those who wish to tweak the implementations, it offers support for Java, R, Python, and Scala. In addition to its native processing engine, which is illustrated in Fig. 3d, they have also released a project called Sparkling Water, shown in Fig. 3e, which integrates Spark and Spark Streaming into their platform. This is only supported in version 3.0. Additional efforts have been made towards integration with Storm for real-time streaming. H_2O's engine processes data completely in-memory using multiple execution methods, depending on what is best for the algorithm used. The general approach used is Distributed Fork/Join, a divide-and-conquer technique, which is reliable and suitable for massively parallel tasks. This is a method which breaks up a job into smaller jobs which run in parallel, resulting in dynamic fine-grain load balancing for MapReduce jobs as well as graphs and streams. They claim to be the fastest execution engine, but as of the time of this writing, no academic studies have been published which verify or refute these claims and further research is needed in this area.

Machine learning toolkits

To perform machine learning tasks in Hadoop, one does not need a special platform or library. A person with programming skills may interact directly with any of the above platforms to roll their own code and many choose to go this route. A variety of machine

learning toolkits have been created to facilitate the learning process but many researchers and practitioners reject them for various reasons, most often because they lack needed features or are difficult to integrate into an existing environment. One issue is that machine learning is a broad field of study and many of the available toolkits lack important functionality. Another problem is that without true expertise in the areas of programming and system architecture, many people lack a full understanding of what the various platforms are capable of. This is exacerbated by the fact that there has been little comprehensive research into many popular frameworks. Research moves much slower than development, so often by the time information becomes abundant, the community has already moved on to different tools. However, distributed learning algorithms are not trivial to implement, and those who do not wish to reinvent the wheel may find that they can save themselves significant effort by using or extending existing implementations. Table 2 provides an overview of four of the more comprehensive machine learning packages that run on Hadoop. Only distributed algorithms are listed in this table. Mahout includes several implementations that are not distributed and therefore not included, but they are discussed in the "Evaluation of machine learning tools" section.

Selection criteria

In a research or production environment, the choice of machine learning packages or specific algorithms will come down to a variety of different factors, mostly dependent on the needs of the specific group or project. A number of authors have tackled this subject, including [87–89]. Based in part on these studies, we offer a list of important considerations for evaluation of machine learning tools. These are presented in no particular order, since the prioritization of these factors will be dependent on particular use cases.

- *Scalability* This should be considered with regards to both the size and complexity of the data. One should consider what their data looks like now, as well as what data they might be working with in the future, in order to determine if a particular toolkit will be appropriate. Scalability should be looked at in both directions, as some of the best tools for big data perform poorly on small data, and vice versa. This is also true for other data characteristics, such as dimensionality.
- *Speed* The biggest factor affecting speed is which processing platform the library or algorithm is running on rather than the library or algorithm itself. However, some libraries are tied to specific platforms, so this is still an important consideration when selecting ML tools. As noted in the previous section, speed may not be important for every project. If models do not require frequent updating, a batch system may be preferred for its simplicity, but for models that are updated often, this may be a crucial concern.
- *Coverage* This refers to the range of options contained in the toolkit in terms of different classes of machine learning as well as variety of implementations in each class. None of the available tools for big data provide a selection as comprehensive as some non-distributed frameworks such as Weka, but their scope may range from only a few algorithms to around two dozen. As many of the tools are difficult to set up and learn, it is important to consider future needs as well as current.

Table 2 Overview of machine learning toolkits

	Mahout	MLlib	H_2O	SAMOA
Interface language	Java	Java, Python, Scala	Java, Python, R, Scala	Java
Associated platform	MapReduce, spark (H2O and flink in progress)	Spark, H_2O	H2O, Spark, MapReduce	Storm, S4, Samza
Current version (as of June 1, 2015)	0.10.1	1.3.1	3.0.0.12	0.2.0
Graphical user interface	–	–	✓	–
Classification and regression algorithms				
Decision tree	–	✓	–	✓[a]
Logistic regression	✓[b]	✓[a]	✓	–
Naïve Bayes	✓	✓	✓	–
Support vector machine	–	✓	–	–
Gradient boosted trees	–	✓	✓	–
Random forest	✓	✓	✓	–
Adaptive model rules	–	–	–	✓[a]
Generalized linear model	–	–	✓	–
Linear regression	–	✓[a]	✓	–
Clustering algorithms				
k-Means	✓	✓	✓	–
Fuzzy k-means	✓	–	–	–
Streaming k-means	✓	✓[a]	–	–
Power iteration	–	✓	–	–
Spectral clustering	✓	–	–	–
CluStream	–	–	–	✓[a]
Collaborative filtering (cf) algorithms				
User-based CF	✓	–	–	–
Item-based CF	✓	–	–	–
Alternating least squares	✓	✓	–	–
Dimensionality reduction and feature selection tools				
Principal component analysis	✓	✓	✓	–
QR decomposition	✓	–	–	–
Singular value decomposition	✓	✓	–	–
Chi squared	✓	–	–	–
Additional algorithms				
Association rule learning	✓	✓	–	✓[a]
Deep learning	–	–	✓	–
Topic modeling	✓	✓	–	–

[a] Real-time streaming implementation

[b] Single machine, trained using Stochastic gradient descent

- *Usability* In this case, one must weigh their project goals against the skills and expertise of their group. Usability may be considered in terms of initial setup, ongoing maintenance, programming languages available, user interface available, amount

of documentation, or availability of a knowledgeable user community. If there is an existing analytics workflow, one should consider how well the tool can be integrated into this.

- *Extensibility* Machine learning tasks are rarely one-size fits all. Whether it's something as simple as setting the value of k in a k-means clustering task [90], or building an ensemble of learners [91], most jobs will require some amount of parameter tuning before a model is deployed. The implementations included in the various tools are often used as building blocks towards new platforms or systems, so it is important to evaluate them in terms of how well they are able to fulfill this role.

Evaluation of machine learning tools

This section provides an in-depth look at the strengths and weaknesses of the various machine learning tools for Hadoop. Published literature using related tools is reviewed here if it is available. For a complete look at how the tools and engines fit together, Fig. 4 illustrates the relationships between processing engines, machine learning frameworks, and the algorithms they implement.

Mahout

Mahout is one of the more well-known tools for ML. It is known for having a wide selection of robust algorithms, but with inefficient runtimes due to the slow MapReduce engine. In April 2015, Mahout 0.9 was updated to 0.10.0, marking something of a shift in the project's goals [92]. With this release, the focus is now on a math environment called Samsara, which includes linear algebra, statistical operations, and data structures. The goal of the Mahout-Samsara project is to help users build their own distributed

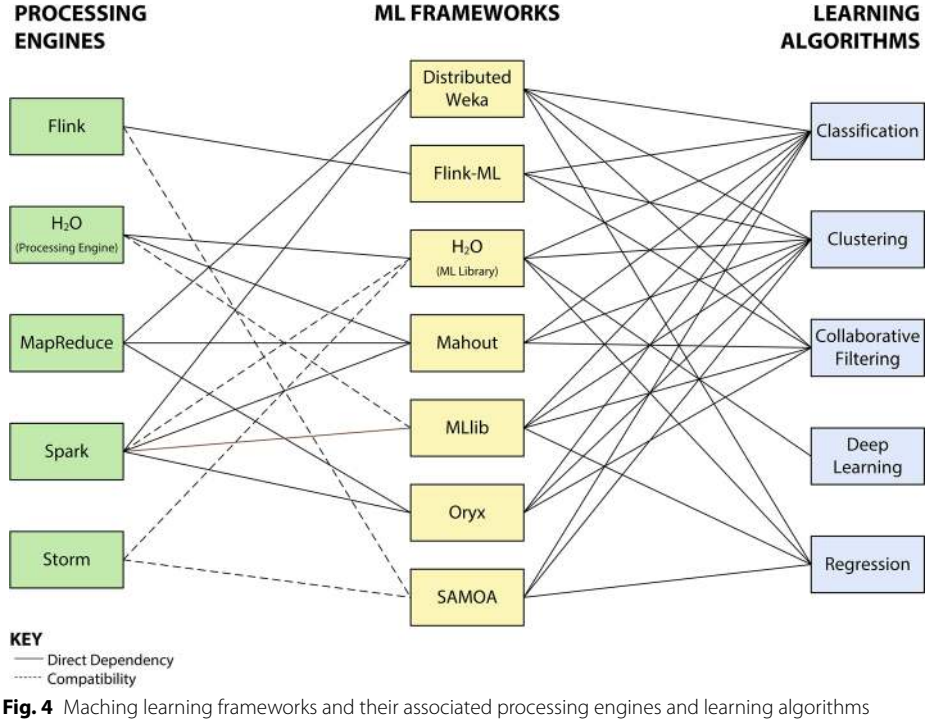

Fig. 4 Maching learning frameworks and their associated processing engines and learning algorithms

algorithms, rather than simply a library of already-written implementations. They still offer a comprehensive suite of algorithms for MapReduce and many have been optimized for Spark as well. Integrations with H_2O and Flink are currently in development. This version is very new, so there is no published literature on it at the time of this writing other than the initial announcement by the developer team introducing the new features. Because most of the old algorithm implementations are still included, the rest of this section focuses on versions 0.9 and earlier.

Among the more commonly cited complaints about Mahout is that it is difficult to set up on an existing Hadoop cluster [93–95]. Additionally, while a lot of documentation exists for Mahout, much of it is outdated and irrelevant to people using the current version. The lack of documentation, a problem common to many machine learning tools, is partially alleviated by an active user community willing and able to help with many issues [96, 97]. One problem with using Mahout in production is that development has moved very slowly; version 0.10.0 was released nearly seven and a half years after the project was initially introduced. The number of active committers is very low, with only a handful of developers making regular commits.

The algorithms included in Mahout focus primarily on classification, clustering and collaborative filtering, and have been shown to scale well as the size of the data increases [98]. Additional tools include topic modeling, dimensionality reduction, text vectorization, similarity measures, a math library, and more. One of Mahout's most commonly cited assets is its extensibility and many have achieved good results by building off of the baseline algorithms [87, 99, 100]. However, in order to take advantage of this flexibility, strong proficiency in Java programming is required [87, 95, 101]. Committer Ted Dunning noted "It's not a product. It's not a package. It's not a service. Batteries are not included [102]." Some researchers have cited difficulty with configuration or with integrating it into an existing environment [33, 93–95]. On the other hand, a number of companies have reported success using Mahout in production. Notable examples include Mendeley [103], LinkedIn [104], and Overstock.com [102], who all use its recommendation tools as part of their big data ecosystems. Overstock even replaced a commercial system with it, saving a significant amount of money in the process.

Classification

Classification algorithms currently offered in Mahout are Logistic Regression, Naïve Bayes, Random Forest, Hidden Markov Models, and Multilayer Perceptron. There has been some additional work towards implementing support vector machines, but this approach is not currently available. Of these algorithms, only Naïve Bayes and Random Forest are parallelized. There are no studies that we are aware of that have utilized Mahout's implementations of Hidden Markov Models or Multilayer Perceptron. This is most likely due to the fact that they are not parallelized.

Logistic Regression is trained via Stochastic Gradient Descent (SGD), negating the need for parallelization [105]. This implementation is favorable due to speed and robustness in the face of new data, but the lack of parallelization may be the reason its use seems to be sporadic. In benchmark tests among different tools [106], Mahout's implementation of Logistic Regression with SGD was left out of that test entirely. The authors stated that its implementation is too communication intensive and to include it in the

comparison would not be fair to Mahout. A different study [107] found Mahout's implementation to be particularly slow in processing sparse data, and it did not fare much better on dense data. Peng et al. [108] compared different approaches to Logistic Regression and noted that Mahout's had poor precision, particularly on imbalanced data, but that it has good scalability and is an example of a sequential algorithm that can train massive data in acceptable time. They recommend Mahout for situations when only a single machine with limited memory is available.

Mahout's implementation of Naïve Bayes is based on [109]. Though this algorithm's (often inaccurate) assumptions of independence may seem counterintuitive, it generally performs quite well in real-world situations, but the performance begins to decline when working with data that is highly imbalanced or dependent [110]. For this reason, Mahout also includes an implementation for Complementary Naïve Bayes, which bases the predictions for class C on the samples belonging to C' (all classes other than the one we are predicting). Both versions have some overhead for training, due to their parallel execution, so they are most effective when using very large data and not recommended for smaller datasets [111]. Due to its simplicity, reliability, and ease of use, Mahout's implementation of Naïve Bayes is often a go-to learner for those looking to demonstrate or test other parts of their system. For example, it was used by [112–114] to test and demonstrate the capability of new data processing engines.

Random Forest is also popular due to its high accuracy and time efficiency [115]. In terms of scalability, Racette et al. [116] compared it with R to predict global conflicts. R crashed when attempting to build more than 250 trees but they were able to use Mahout to build 10,000 on a single node without problems. It should be noted that this is an unfair comparison since R is not built for computation of big data. More research is needed to compare Mahout with similar tools to itself, such as the ones discussed in this paper. Mahout's Random Forest implementation has been applied across many different application domains, but has been used particularly often in healthcare-related studies. It has been used to predict the severity of motor neuron disease [117], identify high risk patients [118], and predict the risk of readmission for congestive heart failure patients [119]. It has also been used as part of a general predictive healthcare analytics framework.

Many studies have described building frameworks for practical machine learning using Mahout. One example is at Honeywell [120], where they built a cloud computing platform combining HBase, Mahout, other analytics tools, and a web interface. In it, they utilized Mahout's Random Forest and Naïve Bayes algorithms to predict events and failures in auxiliary power units before they cause operational interrupts. This system was able to increase predictive accuracy for auto-shutdown events by a factor of more than three.

Clustering

Mahout's library uses several variations of the popular k-Means Clustering algorithm, including the traditional k-Means, Fuzzy k-Means, and Streaming k-Means. Spectral Clustering is also supported. Esteves et al. [101] looked at the performance of Mahout's implementation of k-Means on various sizes of input files and confirmed that Mahout scales very well to larger datasets, but probably would not be a good choice for smaller

ones. Esteves and Rong [121] compared the speed and quality of the traditional and fuzzy implementations of this algorithm by clustering Wikipedia articles. Most experiments were performed 10 times each and they noticed large variations in execution time for the different runs. On average, fuzzy c-means converged faster with less iterations, but k-means was faster on some runs, showing how dependent these measures are on the initial seeding of the centroids. This issue of unpredictable results due to the initial centroid placement has also been noted by [122]. While they initially expected better results from fuzzy k-means due to the inclusion of overlaps, their observed results showed the opposite—centroids were too close together, so the majority of the samples had membership in all clusters. This was attributed to the high dimensionality of the feature vectors, because when they normalized the features during pre-processing it failed to show many of the characteristics that could have better divided the data. In general, k-means produced much more meaningful results than fuzzy k-means, leading to the conclusion that fuzzy k-means is not advised for datasets with high levels of noise.

To the best of our knowledge, no research on Mahout's streaming k-means algorithm has been published. However, Dan Filimon, the designer of Mahout's implementation, presented it at Berlin Buzzwords 2013 [122], and reported his own results from comparisons of k-means and streaming k-means. He found the quality of the clusters formed by each to be comparable, but the streaming implementation was significantly faster. In creating 20 clusters, he found the streaming implementation to be 2.4 times faster, and after going up to 1000 clusters, it finished 8 times faster. To the best of our knowledge this has not yet been independently verified.

In a comparison study of different Spectral Clustering algorithms, Gao et al. [123] noted that Mahout's standard implementation did not scale to datasets larger than 2^{15} instances. However, after some modifications and the addition of local sensitivity hashing as a preprocessing step, they were able to achieve processing times more than an order of magnitude faster than any other implementation.

Collaborative filtering

Mahout is probably the best-known framework for collaborative filtering tools, also known as recommendation engines. The selection of tools they offer in this area is far more robust than what is offered in any of the other toolkits. Currently, Mahout offers implementations for user-based recommendations, item-based based recommendations, and several variations of matrix factorization. Mahout also offers a number of tools to compute the similarity measures of any of the above recommenders. These include Pearson correlation, Euclidean distance, Cosine similarity, Tanimoto coefficient, log-likelihood, and others. Evaluation of recommenders is tricky, due to the nature of the task, so if testing a system using feedback from actual users is not possible, Mahout provides a hold-out test, in which a portion of the training data is set aside to use for testing [111].

In a comparison between MLI (an API for distributed machine learning built on Spark), GraphLab, Mahout, and MATLAB of collaborative filtering with alternating least squares [106], it was observed that Mahout's recommenders are fairly easy to set up on an existing cluster but significant tuning is required to run them effectively on large datasets while ensuring good performance. Mahout's implementation had significantly

more lines of code, was slower on all datasets, and didn't scale as well as any of the others. There is some disagreement on scalability though, as "proven scalability" is cited by [124] as one of Mahout's strengths in this area. However, the same study criticized its lack of built-in hybrid recommenders, support for groups, or context awareness.

Some of these issues are likely indicative of a problem with MapReduce rather than the Mahout code. Iterative algorithms are known not to run well on Hadoop/MapReduce because of the need to access data and variables from disks [125]. However, to the best of our knowledge, Mahout's implementation has not been formally benchmarked on a memory-based platform such as Spark or H_2O, and until this happens, it is difficult to properly judge the scalability of the actual code. Also not explored in these studies is the fact that Mahout's recommenders are designed to be highly extensible, so good or bad performance on a baseline implementation may not be indicative of what the tools are capable of when tuned properly for a specific application or dataset. A study designed to compare the baseline algorithms offered out-of-box to various customized versions found significant improvement in performance when employing new techniques such as significance weighting and mean-centered prediction [98]. Mendeley, an application used by students and researchers to organize and share reference material, leverages Mahout's collaborative filtering algorithms to help users discover new articles and papers [103]. When evaluating the item-based recommender, they determined it was efficient but could be improved upon. By allocating additional mappers and reducers and using an appropriate partitioner, they were able to reduce processing time by 63 %.

Said and Bellogín [126] stress the importance of having clear guidelines in comparison of recommender systems to allow for reproducibility and comparison of results. They note that a true comparison of results from different recommender systems is difficult due to the many different designs available, as well as myriad differences in implementation strategies from modification and tuning. With that in mind, they looked at three popular frameworks which include Mahout, LensKit, and MyMediaLite, and compared various research papers based on the reproducibility of the results. For each framework, a controlled evaluation was performed as well as one using the framework's internal evaluation methods. The results of the controlled evaluation showed Mahout's performance to be consistent and fast for all algorithms, but its performance was still slightly slower than Lenskit. The authors note that these results are specific to the datasets used and may differ when running on datasets with different characteristics. When considering root-mean-squared error, Mahout was the worst of the three. This study only used non-distributed implementations from Mahout for fair comparisons, so their observations should not be extrapolated to the distributed implementations offered.

MLlib

MLlib covers the same range of learning categories as Mahout, and also adds regression models, which Mahout lacks. They also have algorithms for topic modeling and frequent pattern mining. Additional tools include dimensionality reduction, feature extraction and transformation, optimization, and basic statistics. In general, MLlib's reliance on Spark's iterative batch and streaming approaches, as well as its use of in-memory computation, enable jobs to run significantly faster than those using Mahout [88, 127].

However, the fact that it is tied to Spark may present a problem for those who perform machine learning on multiple platforms [128].

MLlib is still relatively young compared to Mahout. As such, there is not currently an abundance of published case studies that have used this library, and there is very little research providing meaningful evaluation. The research that has been published indicates it is considered to be a relatively easy library to set up and run [129], helped in large part by the fact that it ships as part of its processing engine, thus avoiding some of the configuration issues people have reported with Mahout. Zheng and Dagnino [127] note that the underlying optimization primitives make it easy to extend existing algorithms or write new parallel implementations. There have been questions raised about performance and reliability of their algorithms and more research needs to be done in this area [127]. Efficiency issues have also been noted due to slow convergence requiring a large number of iterations as well as high communication costs [130, 131]. Other studies [129, 131, 132] have discussed problems in how the implemented tools handle less-than-ideal data such as very low or very high dimensional vectors.

The documentation is thorough, but the user community is not nearly as active as the community developing for it. This issue is expected to improve as more people are migrating from MapReduce to Spark. The large and active group of developers means that many complaints are fixed before they are even published. Notable examples of companies that use MLlib in production are OpenTable and Spotify [133], both for their recommendation engines.

Algorithms

For most of its lifetime, much of the effort that has gone into this project has focused on developing the Spark platform rather than expanding libraries. The focus has been on developing a few widely-understood algorithms well rather than the grab-bag approach taken by Mahout. Recently, however, they have upped their efforts in this area and expanded their offerings. One would expect them to stay on this track given the increasingly widespread adoption of the Spark platform. For classification, they have Support Vector Machines, Logistic Regression, Naïve Bayes, Decision Trees, Random Forest, and Gradient-Boosted Trees. Clustering algorithms include k-Means, Gaussian Mixture, and Power Iteration Clustering. They offer implementations for Linear Regression and Isotonic Regression, and one collaborative filtering algorithm using Alternating Least Squares.

For online learning, streaming versions of Logistic Regression, Linear Regression, and k-Means Clustering are included. For all other algorithms, models can be learned offline using historic data and applied online to new streaming data. MLlib includes APIs for development in Scala, Java and Python, but not every tool is available in all languages.

ML pipelines

As we have discussed throughout this paper, building machine learning pipelines can be a difficult task, particularly when working with a combination of disparate tools. Spark ML, a set of uniform APIs for creation and tuning of pipelines was introduced in version 1.2 to address these issues, making it easier to combine multiple algorithms into one workflow. This package includes tools for dataset transformations and combining

algorithms. It works by representing a pipeline as a sequence of dataset transformations. An easy example of this is to think of a learner which transforms a DataFrame with features into one with predictions. This package is designed to handle all steps of the learning process, starting with importing data from a source, to extracting features, and training and evaluating models.

MLbase

Though it is not currently available, there has been ongoing research and development at Berkeley's AMP lab on a platform called MLbase, which wraps MLlib, Spark, and other projects to make machine learning on data sets of all sizes accessible to a broader range of users [134–136]. In addition to MLlib and Spark, the other core components are MLI, an API for feature extraction and algorithm development, and ML Optimizer, which automates the tuning of hyperparameters.

A new component called TuPAQ (Training-supported Predictive Analytic Query Planner) [137] was recently introduced, which builds on the initial idea of ML Optimizer. TuPAQ serves as a query interface that allows a user to input high level queries in a declarative language and then selects for them the best model and parameters. One of the goals in the development of MLbase was to make machine learning accessible for the non-expert. TuPAQ is an important step toward this goal as it pushes hyperparameter tuning, feature selection and algorithm selection down into the system. Another stated goal was to also make this system valuable for the experts. In this architecture, they are able to work directly with MLI and MLlib to develop their own implementations or tune existing ones, effectively skipping the PAQ step. They have also recently introduced a new tool, GHOSTFACE, which aims to automatically perform model selection for the user [138].

H_2O

Out of all of the tools discussed in this paper, H_2O is the only one that can be considered a product, rather than a project. While they offer an enterprise edition with two tiers of support, nearly all of their offerings are available open source as well and can be used without the purchase of a license. The most notable features of this product are that it provides a graphical user interface (GUI), and numerous tools for deep neural networks. Deep learning has shown enormous promise for many areas of machine learning, making it an important feature of H_2O. [139]. There is another company offering open source implementations for deep learning, Deeplearning4j [140], but it is targeted towards business instead of research, whereas H_2O targets both. Additionally, Deeplearning4j's singular focus is on deep learning, so it doesn't offer any of the other types of ML tools that are in H_2O's library. There are also other options for tools with a GUI, such as Weka, KNIME [141], or RapidMiner [142], but none of them offer a comprehensive open source machine learning toolkit that is suitable for big data.

Programming in H_2O is possible with Java, Python, R and Scala. Users without programming expertise can still utilize this tool via the web-based UI. Because H_2O comes as a package with many of the configurations already tuned, set up is easy, requiring less of a learning curve than most other free options. While H_2O maintains their own

processing engine, they also offer integrations that allow use of their models on Spark and Storm.

As of May 2015, the machine learning tools offered cover a range of tasks, including classification, clustering, generalized linear models, statistical analysis, ensembles, optimization tools, data preprocessing options and deep neural networks [143]. On their roadmap for future implementation are additional algorithms and tools from these categories as well as recommendation and time-series. Additionally, they offer seamless integration with R and R Studio, as well as Sparkling Water for integration with Spark and MLlib. An integration with Mahout is currently in the works as well. They offer thorough documentation and their staff is very communicative, quickly answering questions in their user group and around the web.

To the best of our knowledge, only a single paper has been published so far that used this product in a study [144]. Their work found H_2O to be an effective and reliable tool for analyzing sensor data and predicting missing measurements. This study looked at Gradient Boosted Model (GBM) and Generalized Linear Model (GLM) and found them to be comparable when used for classification, though GBM produced better accuracy when used for regression. More independent research is needed to properly evaluate the speed, performance, and reliability of this tool. One notable example of a company using H_2O in production is ShareThis, who uses predictive modeling to maximize advertising ROI (return on investment).

SAMOA

SAMOA, a platform for machine learning from streaming data, was originally developed at Yahoo! Labs in Barcelona in 2013 and has been part of the Apache incubator since late 2014. Its name stands for Scalable Advanced Massive Online Analysis. It is a flexible framework that can be run locally or on one of a few stream processing engines, including Storm, S4, and Samza. This is done through a minimal API designed for a general distributed stream processing engine which allows users to easily write bindings to port SAMOA to new stream processors [145].

Though they currently offer far fewer algorithms, they like to call themselves "Mahout for streaming." SAMOA's algorithms are represented as directed graphs, referred to as topologies (borrowing terminology from Storm). The algorithms implemented so far can be used for classification, clustering, regression, and frequent pattern mining, along with boosting, and bagging for ensemble creation. Additionally, there is a common platform provided for their implementations, as well as a framework for the user to write their own distributed streaming algorithms. It does not yet have an active community, but it offers thorough documentation.

This platform is meant for users with very big data that is constantly being updated. Streaming models are for projects aimed at finding out what is happening *right now,* and feedback occurs in real-time. SAMOA was designed with the goals of flexibility in updating the library (developing new implementations as well as reusing existing ones from other frameworks), scalability in its handling of an increasing amount of data, and extensibility in terms of the APIs described above [146]. Internal tests have resulted in high speed and accuracy. Severien [147] evaluated the flexibility of the platform by implementing CluStream, a stream clustering algorithm, using SAMOA's API

for creating data flow topologies. One of the benefits to this API is that it allows the user to implement new algorithms that will automatically be able to run on any processing engine that is able to plug into SAMOA. While more research is needed to fully evaluate these tools, results from this study indicated that it is a flexible platform with the ability to scale up to larger workloads and shows good potential for use in a large-scale distributed environment. This implementation is now part of SAMOA's library. Romsaiyud [148] implemented a topic modeling algorithm and compared experimental results using both SAMOA and MOA. Results indicated significantly higher throughput on SAMOA and showed the framework to be robust and stable.

So far, there are only a few learning tools implemented in SAMOA, but they cover the many of the common ML tasks. For clustering, they now offer CluStream, and for classification there is the Vertical Hoeffding Tree, which utilizes vertical parallelism on top of the Very Fast Decision Tree, or Hoeffding Tree. This is the standard decision tree algorithm for streaming classification tasks. Regression can be accomplished through the Adaptive Model Rules Regressor, which includes implementations for both vertical and horizontal parallelism. The library also includes Distributed Stream Frequent Itemset Mining, which is based on PARMA [149]. Prequential Evaluation is available as well, which enables measurement of model accuracy, either from the start or based on a sliding window of recent instances. Bagging, Adaptive Bagging, and Boosting can be used to create ensembles of classifiers. For additional learning algorithms, there is a plugin available called SAMOA-MOA [150] which allows the use of MOA classifiers and clustering algorithms inside the SAMOA platform. However, it is important to keep in mind that this does not change the underlying implementations of MOA's algorithms, which are not distributed. SAMOA is a very young project and new tools are continually being developed to expand the library [151].

There is not a great deal of independent research on this platform, though that is likely to change as SAMOA becomes more well-known and online learning becomes more widely used. Rahnama [152] used SAMOA and Storm to perform sentiment analysis on real-time Twitter streams. To round out the project, he developed an open source Java library called Sentinel [153] which includes tools for stream reading, data pre-processing, query response, feature selection and frequent itemset mining tailored to sentiment analysis tasks on Twitter data. Analysis was carried out through an ensemble of Vertical Hoeffding Trees using Adaptive Bagging. Other projects have leveraged SAMOA as part of frameworks for efficiently finding top-k items [154] and for recognition of internet traffic patterns [155].

Comparison of major machine learning frameworks

Using the evaluation standards that were discussed in the previous section, we have assigned a rating to each of the four major frameworks based on the available literature.[2] A comparison of the frameworks is shown in Fig. 5. The figure is meant to be viewed as a comparative ranking of the tools displayed in the figure, with each tool ranked according to the selection criteria that have been outlined in this paper. These are relative ratings based on comprehensive literature and online documentation review, not our own

[2] There is not yet any literature available on the newest version of Mahout, so these ratings reflect the results of studies using versions 0.9 and earlier.

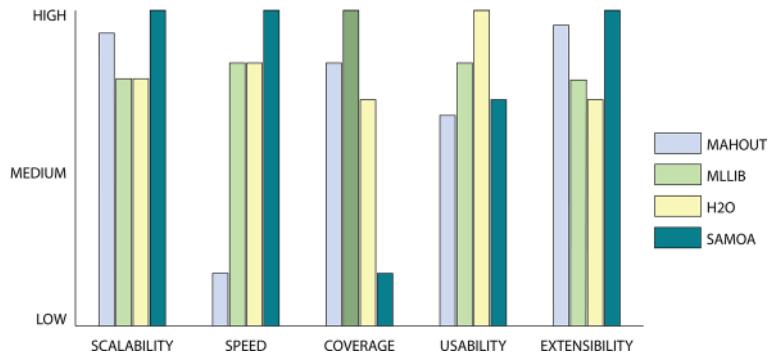

Fig. 5 Comparison of machine learning frameworks

experimental results. Future work will include quantitative comparisons of these tools based on formally defined criteria, but for this survey, we provide qualitative rankings based on our exposure to each tool and related works. While a number of other tools are available, some of which will be discussed below, there is not yet enough literature in order to properly evaluate them.

As stated previously, the choice of ML framework will be largely application and user-specific. MLlib and H_2O are very good options for general needs; each is fast, scalable to different dataset sizes, and has a fairly diverse selection of algorithms. MLlib offers a better selection for most areas of machine learning, but H_2O is the only tool that has solutions for deep learning. In terms of usability, both have APIs for programming in multiple languages, and H_2O also offers a GUI, making it easier to use for those without a high level of programming expertise.

SAMOA and Mahout (through the new Mahout-Samsara) both focus on offering platforms through which the user can create his or her own implementations of learning algorithms, so if none of the libraries contain needed algorithms or if future extensibility is an important factor, one of the two would be advisable. SAMOA allows the user to create implementations of streaming algorithms, while Mahout-Samsara can be used for batch implementations. SAMOA is the only one which is designed for true real-time streaming, making it the fastest and most scalable option.

Additional tools

The learning frameworks discussed in this section have been chosen due to their widespread use or versatility with respect to implemented tools on a range of applications. This is by no means a comprehensive list of all open source learning tools for Hadoop and there are additional frameworks which show promise for large-scale machine learning tasks. This section provides a brief overview of several such frameworks that were left out of the main discussion either because they aren't very versatile or there is a lack of published research.

Flink-ML is a machine learning library currently in development for the Flink platform. It supports implementations of Logistic Regression, k-Means Clustering, and Alternating Least Squares (ALS) for recommendation. It also supports Mahout's DSL (Domain Specific Language) for linear algebra which can be used for optimization of learning algorithms, and plans are underway to implement pre- and post-processing

tools. For graph processing, they have the Gelly API which provides methods for vertex-centric iterations.

Weka began including wrappers for distributed processing on Hadoop in version 3.7 [156]. *distributedWekaBase* is a package that provides map and reduce tasks that are not tied to a specific platform, and could potentially be used to add new platforms in the future. *distributedWekaHadoop* provides utilities specific to Hadoop, including loader and savers for HDFS and configuration for Hadoop tasks. This is only available for Hadoop version 1.1.2, which was pre-YARN, meaning it will only run on MapReduce. One reason for Weka's appeal is the vast amount of tools in their library. While their classifiers and regressors are able to be used on Hadoop, most of them are not able to be parallelized. These algorithms are essentially trained as ensembles, in which smaller data subsets are trained individually but instead of merging them into a final model during the reduce phase, they are combined using voting techniques. This integration was introduced in 2013, but has not seen widespread adoption. This may be due to the lack of parallel algorithms. A similar package for Spark, *distributedWekaSpark* was introduced in March 2015. Currently it can only accept input from .csv files.

Oryx (formerly Myrrix) [157] does not offer a broad selection of algorithms, but still covers the major areas of classification, clustering, and collaborative filtering for real-time large scale ML. Its architecture consists of a computation layer which builds models and a serving layer which exposes a REST API that can be accessed from a browser or any tool that is able to make HTTP requests. It offers implementations for k-Means++, Random Forests, and Matrix Factorization based on a variant of Alternating Least Squares for Collaborative Filtering. Oryx implements a lambda architecture so models can be updated and queried in real time. While documentation is very limited, it is said to be easy to configure and get running [93]. It currently runs on MapReduce, but version 2 [158], which is in development, is built on Spark and Kafka.

Vowpal Wabbit [159] is a fast online learner originally developed at Yahoo! Research Labs and currently sponsored by Microsoft Research. It is designed for terafeature datasets and offers support for classification, matrix factorization, topic modeling, and optimization. It is extremely fast and efficient due to its use of feature hashing. However, documentation is lacking, and while it is popular among researchers, it has been said to be difficult to integrate into a production environment [111].

Suggestions for future work

Thus far, most of the research in the area of machine learning for big data has focused on processing paradigms and algorithm implementation and optimization. Largely ignored in the research is the development of tools for the data itself, specifically for preprocessing techniques. We argue that while each of the above tools has their advantages and drawbacks, all of them could be improved with easier to use, and more efficient tools for dealing with problems inherent to big data. Some of these issues include:

- *Mislabeled data* As data grows, the likelihood of having mislabeled instances grows as well. When dealing with millions of instances, it is not possible to efficiently check whether all of the training data is properly labeled, and training models on incorrect data will lead to less accuracy [160]. While some big data algorithms include mech-

anisms to handle this problem, it may be helpful to have tools that offer solutions before classification begins.

- *Missing values* Similar to mislabeled data, missing values also lead to less robust, inaccurate models, particularly with clustering and collaborative filtering algorithms which depend on similarity computations. This issue is generally solved either through imputation techniques or by removing the example completely [161].
- *Noise* Noisy data refers to data that is irrelevant or meaningless. This can lead to models that suffer from overfitting. Clustering or similarity measures can help identify noisy data points, but toolkits lack algorithms which are specifically optimized for this task [162].
- *High dimensionality* This occurs when the feature-to-instance ratio is very large and is a common characteristic of big data. Algorithms for dimensionality reduction, most commonly Principal Component Analysis (PCA), are included in most toolkits. Feature selection is a well-known method to handle high-dimensionality [163], and PCA is just one option of many that could be included.
- *Imbalance* In classification problems, imbalanced training data (when there are many more instances in one or more classes than in others) can lead to weak learners. This is typically alleviated by using data sampling techniques [164].

A number of studies have suggested that many of the learning algorithms implemented in these tools do not stand up well to these kinds of problems [121, 165 166]. There are well-used and often simple techniques to combat these issues. Some of them are implemented in various machine learning packages, but many are not. And when they are included, they can be difficult to find or use. Most libraries have tools addressing dimensionality reduction, but solutions to the other problems listed have not been widely implemented. There is a huge need for effective tools, particularly in production environments [167], making this a valuable problem to address. There are a number of studies which examine solutions to these issues for the platforms discussed in this paper [99, 168], but none within the context of a machine learning toolkit. Any of the platforms discussed in this paper could become much more robust with the addition of some of these tools.

Conclusion

Current trends in technology, such as increased adoption of wearable computers and other Internet of Things (IoT) devices, are allowing for unprecedented access to massive amounts of heterogeneous data. Efficient learning from this data often requires complex architectures that utilize a combination of tools and techniques for collection, storage, processing, and analysis [169]. Putting together such an architecture would be extremely difficult even if there were limited options from which to choose. The open-source data science community is prolific, resulting in many options and many more decisions to be made.

As machine learning concepts are being increasingly adopted in research and production settings, the need for tools to facilitate learning tasks is becoming more important. Many of these tools are very young, and more research is needed to properly benchmark and evaluate all of the different options. Areas where this research is insufficient have

been noted throughout this paper. We discussed the Hadoop ecosystem and a number of tools that are a part of it in order to provide context to how machine learning fits into an analytics environment. Three major approaches to processing (batch, iterative batch, and real-time streaming) were described and projects using each of them were presented and compared. Additionally, a list of criteria for evaluation and selection of machine learning frameworks was presented along with an in-depth look at both widely used and up-and-coming projects, with a discussion of their advantages and drawbacks.

We chose to focus the bulk of our research on processing engines and machine learning frameworks because those are the two most important types of tools in an ML pipeline. We chose projects in the Hadoop ecosystem for a number of reasons. First, they are among the most innovative we have seen. Additionally, there are few end-to-end services out there for machine learning and Hadoop projects tend to be designed with the intention of connecting with ones that already exist in this group. Finally, Apache projects tend to draw large numbers of active users who are helpful when problems arise.

The choice of tools will largely depend on the applications they are being used for as well as user preferences. For example, Mahout, MLlib, Flink-ML, and Oryx include options for recommendations, so if the intended application is an e-commerce site or social network, one may wish to choose from them for features such as item or user suggestions. Social media or IoT data may require real-time results, necessitating the use of Storm or Flink along with their associated ML libraries. Other domains such as healthcare often produce disparate datasets that may require a mix of batch and streaming processing, in which case Flink, Oryx, or Spark would be the best choice.

In this paper, we examined five processing platforms. MapReduce, once the de facto standard for big data projects, is becoming outmoded in the machine learning community and is not recommended for the majority of applications due to its slowness and lack of support for iterative algorithms. Spark is seen by many as a natural successor. It is based in MapReduce so the transition is not difficult, but it offers support for iterative tasks and is able to support all of the machine learning libraries that MapReduce does plus others. However, if real-time solutions are of importance, one may wish to consider Storm or Flink instead, since they offer true stream processing while Spark's use of microbatch streaming may have a small lag associated with it in receiving results. Flink offers the best of both worlds in this regard, with a combination of batch and true stream processing, but it is a very young project and needs more research into its viability. Additionally, it does not currently support nearly as many machine learning solutions as the other platforms. H_2O is the only end-to-end system discussed in this paper and offers two features not present in other systems, which are a graphical user interface, and support for deep learning. Additionally, it supports as many or more machine learning tools than any of the other engines we studied. Like Flink, there is very little research on H_2O, so more is needed for a proper evaluation.

No distributed ML libraries have the same amount of options as some of the non-distributed tools such as Weka because not every algorithm lends itself well to parallelization. Mahout and MLlib are the most well-rounded big data libraries in terms of algorithm coverage and both work with Spark and H_2O. MLlib has a wider overall selection of algorithms and a larger and more dedicated team working on it, but is young and largely unproven. Mahout includes the most options for recommendation and has

more maturity than the others. While Mahout was falling out of favor due to its reliance of MapReduce but this may change due to modifications made in the newest version. Now that it is focused more on the math needed for users to create their own algorithm implementations, we can conceive of situations in which a user may wish to be familiar with and utilize both Mahout and MLlib. SAMOA, like the new version of Mahout, has a focus on giving users the necessary tools to create their own implementations, the small amount of research in this area suggests it to be a viable option for stream processing. Real-time learning is increasing in popularity and we expect the amount of options available for it to increase as well.

Authors' contributions

SL performed the primary literature review and analysis for this work, and also drafted the manuscript. ANR and TH worked with SL to provide additional analysis and develop the paper's framework and focus. TMK introduced this topic to SL, ANR, and TH, and coordinated the other authors to complete and finalize this work. All authors read and approved the final manuscript.

Acknowledgements

The authors gratefully acknowledge partial support by the National Science Foundation, under grant number CNS-1427536. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Competing interests

The authors declare that they have no competing interests.

Received: 9 June 2015 Accepted: 20 October 2015
Published online: 05 November 2015

References

1. International Data Corporation. Digital Universe Study. 2014. http://www.emc.com/leadership/digital-universe/index.htm. Accessed 1 Jun 2015.
2. Ancestry.com Fact Sheet. http://corporate.ancestry.com/press/company-facts/. Accessed 1 Jun 2015.
3. The R Project for Statistical Computing. http://www.r-project.org/.
4. Weka. http://www.cs.waikato.ac.nz/ml/weka/.
5. Apache Hadoop. https://hadoop.apache.org/.
6. Laney D. 3D data management: controlling data volume, velocity and variety. META Group; 2001.
7. Demchenko Y, Grosso P, de Laat C, Membrey P. Addressing big data issues in scientific data infrastructure. In: 2013 International Conference on Collaboration Technologies and Systems (CTS), San Diego, 2013. IEEE, pp 48–55.
8. Cox M, Ellsworth D. Managing big data for scientific visualization. In: ACM Siggraph '97 course #4 exploring gigabyte datasets in real-time: algorithms, data management, and time-critical design, August, 1997.
9. Bekkerman R, Bilenko M, Langford J. Scaling up machine learning: parallel and distributed approaches. Cambridge: Cambridge University Press; 2011.
10. White T. Hadoop: The Definitive Guide, 3rd edn. Sebastopol, CA:O'Reilly Media, Inc.; 2012.
11. Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, Saha B, Curino C, O'Malley O, Radia S, Reed B, Baldeschwieler E. Apache Hadoop YARN: Yet Another Resource Negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing; 2013.
12. Apache Hadoop 2.7.0 Documentation. http://hadoop.apache.org/docs/current/. Accessed 5 Jan 2015.
13. Cloudera. http://www.cloudera.com/.
14. Hortonworks. http://hortonworks.com/.
15. MapR. https://www.mapr.com.
16. Project Voldemort. http://www.project-voldemort.com/voldemort/.
17. Redis. http://redis.io/.
18. Apache CouchDB. http://couchdb.apache.org/.
19. MongoDB. https://www.mongodb.org/.
20. Apache HBase. http://hbase.apache.org/.
21. Apache Cassandra. http://cassandra.apache.org/.
22. Titan Distributed Graph Database. http://thinkaurelius.github.io/titan/.
23. Neo4j. http://neo4j.com/.
24. OrientDB. http://orientdb.com/orientdb/.
25. Apache Flume. https://flume.apache.org/.
26. Apache Kafka. http://kafka.apache.org/.
27. Apache Sqoop. http://sqoop.apache.org/.
28. Apache Hive. http://hive.apache.org/.
29. Apache Drill. http://drill.apache.org/.

30. Fernández A, del Río S, López V, Bawakid A, del Jesus MJ, Benítez JM, Herrera F. Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. Wiley Interdiscip Rev Data Min Knowl Discov. 2014;4(5):380–409.
31. Cascading. http://www.cascading.org/.
32. Apache Pig. http://pig.apache.org/.
33. Lin J, Kolcz A. Large-scale machine learning at twitter. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data; 2012. pp. 793–804.
34. Apache Tez. http://tez.apache.org/.
35. Apache Oozie Workflow Scheduler for Hadoop. http://oozie.apache.org/.
36. Apache Zookeeper. https://zookeeper.apache.org/.
37. Hue. http://gethue.com/.
38. MOA (Massive Online Analysis). http://moa.cs.waikato.ac.nz/.
39. Hellerstein JM, Schoppmann F, Wang DZ, Fratkin E, Welton C. The MADlib Analytics Library or MAD Skills, the SQL. In: VLDB Endowment; 2012. pp. 1700–11.
40. Dato Core. https://github.com/dato-code/Dato-Core.
41. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings of the 6th Symposium on Operating Systems Design and Implementation; 2004.
42. Dewitt D, Stonebraker M (2008) MapReduce : a major step backwards. Database Column.
43. DeMichillie G. Reimagining developer productivity and data analytics in the cloud—news from Google IO. 2014. http://googlecloudplatform.blogspot.com/2014/06/reimagining-developer-productivity-and-data-analytics-in-the-cloud-news-from-google-io.html. Accessed 5 Jan 2015.
44. Apache Giraph. http://giraph.apache.org/.
45. Apache Hama. https://hama.apache.org/.
46. Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, and Czajkowski G. Pregel: A System for Large-Scale Graph Processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data; 2010. pp. 135–45.
47. Amazon EC2. http://aws.amazon.com/ec2/.
48. Microsoft Azure. http://azure.microsoft.com/.
49. Attenberg J. Conjecture: Scalable Machine Learning in Hadoop with Scalding. 2014. https://codeascraft.com/2014/06/18/conjecture-scalable-machine-learning-in-hadoop-with-scalding/. Accessed 1 Jun 2015.
50. Zaharia M, Chowdhury M, Das T, Dave A. Fast and interactive analytics over Hadoop data with Spark. USENIX Login. 2012;37(4):45–51.
51. Bu Y, Howe B, Balazinska M, Ernst MD. HaLoop: efficient Iterative Data Processing on Large Clusters. Proceedings VLDB Endowment. 2010;3(1):285–96.
52. Jakovits P, Srirama SN. Evaluating MapReduce frameworks for iterative Scientific Computing applications. In: 2014 International Conference on High Performance Computing & Simulation; 2014. pp. 226–33.
53. Spark. https://spark.apache.org/.
54. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster Computing with Working Sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing; 2010.
55. Ni Z. Comparative Evaluation of Spark and Stratosphere. Thesis, KTH Royal Institute of Technology; 2013.
56. Xin R. DataFrames for Large-Scale Data Science. Databricks TechTalk. https://www.youtube.com/watch?v=Hvke1f10dL0 (2015).
57. Sort Benchmark Home Page. http://sortbenchmark.org/. Accessed 1 Jun 2015.
58. Xin R. Spark officially sets a new record in large-scale sorting. 2014. http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html. Accessed 01 Jun 2015.
59. Cai Z, Gao J, Luo S, Perez LL, Vagena Z, Jermaine C. A comparison of platforms for implementing and running very large scale machine learning algorithms. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data (SIGMOD'14); 2014. pp. 1371–82.
60. MLlib. https://spark.apache.org/mllib/.
61. GraphX. https://spark.apache.org/graphx/.
62. Mahout. http://mahout.apache.org/.
63. Zhang H, Tudor BM, Chen G, Ooi BC. Efficient in-memory data management: an Analysis. Proceedings VLDB Endowment. 2014;7(10):6–9.
64. Singh J. Big Data Analytic and Mining with Machine Learning Algorithm. Int J Inform Comput Technol. 2014;4(1):33–40.
65. Ousterhout K, Rasti R, Ratnasamy S, Shenker S, Chun B. Making Sense of Performance in Data Analytics Frameworks. In: Proceedings of the 12th USENIX Symposium. On Networked Systems Design and Implementation (NSDI 15); 2015.
66. Shahrivari S, Jalili S. Beyond batch processing : towards real-time and streaming big data. Computers. 2014;3(4):117–29.
67. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I. Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing. University of California at Berkeley Technical Report No. UCB/EECS-2012-259; 2012.
68. Apache Storm. https://storm.apache.org/.
69. Marz N. History of Apache Storm and lessons learned. 2014. http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html. Accessed 12 Apr 2015.
70. Khudairi S. The Apache Software Foundation Announces Apache™ Storm™ as a Top-Level Project. 2014. https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces64.
71. Lorica B. A real-time processing revival. Radar. 2015. http://radar.oreilly.com/2015/04/a-real-time-processing-revival.html.
72. Apache Thrift. http://thrift.apache.org/.

73. Toshniwal A, Donham J, Bhagat N, Mittal S, Ryaboy D, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M. Storm @Twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data (SIGMOD'14); 2014. pp. 147–56.
74. Gradvohl ALS, Senger H, Arantes L, Sens P. Comparing Distributed Online Stream Processing Systems Considering Fault Tolerance Issues. J Emerg Technol Web Intell. 2014;6(2):174–9.
75. Feng A, Evans R, Dagit D, Roberts N. Storm-yarn. https://github.com/yahoo/storm-yarn.
76. Marz N, Warren J. Big data: principles and best practices of scalable realtime data systems. Manning Publications; 2015.
77. H_2O. http://h2o.ai/.
78. Real-time Predictions with H2O on Storm. https://github.com/h2oai/h2o-training/blob/master/tutorials/streaming/storm/README.md#real-time-predictions-with-h2o-on-storm.
79. Wasson T, Sales AP. Application-Agnostic Streaming Bayesian Inference via Apache Storm. In: The 2014 International Conference on Big Data Analytics; 2014.
80. Merienne P, Trident-ml. http://github.com/pmerienne/trident-ml.
81. Apache Flink. https://flink.apache.org/.
82. Alexandrov A, Bergmann R, Ewen S, Freytag JC, Hueske F, Heise A, Kao O, Leich M, Leser U, Markl V, Naumann F, Peters M, Rheinländer A, Sax MJ, Schelter S, Höger M, Tzoumas K, Warneke D. The Stratosphere platform for big data analytics. VLDB J Int J Very Large Data Bases. 2014;23(6):939–64.
83. Ewen S, Schelter S, Tzoumas K, Warneke D, Markl V. Iterative Parallel Data Processing with Stratosphere : An Inside Look. In: Proceedings of the 2013 International Conference on Management of Data (SIGMOD'13); 2013. pp. 1053–6.
84. Leich M, Adamek J, Schubotz M, Heise A, Rheinländer A, Markl V. Applying Stratosphere for Big Data Analytics. In: 15th Conference on Database Systems for Business, Technology and Web (BTW 2013); 2013. pp. 507–10.
85. Flink-ML. https://github.com/apache/flink/tree/master/flink-staging/flink-ml.
86. Metzger R, Celebi U. Introducing Apache Flink—A new approach to distributed data processing. In: Silicon Valley Hands On Programming Events; 2014.
87. Chalmers S, Bothorel C, Picot-Clemente R. Big Data—State of the Art. Technical Report, Telecom Bretagne, Technical Report; 2013.
88. Singh D, Reddy CK. A survey on platforms for big data analytics. J Big Data. 2014;1:8.
89. Collier K, Carey B, Sautter D, Marjaniemi C. A methodology for evaluating and selecting data mining software. In: Proceedings of the 32nd Annual Hawaii International Conference on Systems sSciences, Maui, HI; 1999. IEEE, pp. 11.
90. Zhong S, Khoshgoftaar TM, Seliya N. Clustering-based network intrusion detection. Int J Reliab Qual Saf Eng. 2007;14(02):169–87.
91. Khoshgoftaar TM, Dittman DJ, Wald R, Awada W. "A review of ensemble classification for dna microarrays data," in Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on. IEEE, 2013, pp. 381–9.
92. Apr 2015—Apache Mahout's next generation version 0.10.0 released. http://mahout.apache.org/. Accessed 16 Apr 2015.
93. Miller J. Recommender System for Animated Video. Issues Inform Syst. 2014;15(2):321–7.
94. Wegener D, Mock M, Adranale D, Wrobel S. Toolkit-Based High-Performance Data Mining of Large Data on MapReduce Clusters. In: 2009 IEEE International Conference on Data Mining Workshops; 2009. pp. 296–301.
95. Zeng C, Jiang Y, Zheng L, Li J, Li L, Li H, Shen C, Zhou W, Li T, Duan B, Lei M, and Wang P. FIU-Miner: A Fast, Integrated, and User-Friendly System for Data Mining in Distributed Environment. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining; 2013. pp. 1506–9.
96. Geng X, Yang Z. Data Mining in Cloud Computing. In: Proceedings of the 2013 International Conference on Information Science and Computer Applications (ISCA 2013); 2013.
97. De Souza RG, Chiky R, Aoul ZK. Open Source Recommendation Systems for Mobile Application. In: Workshop on the Practical Use of Recommender Systems, Algorithms and Technologies (PRSAT 2010); 2010. pp. 55–8.
98. Seminario CE, Wilson DC. Case Study Evaluation of Mahout as a Recommender Platform. In: 6th ACM conference on recommender engines (RecSys 2012); 2012. pp. 45–50.
99. Lemnaru C, Cuibus M, Bona A, Alic A, Potolea R. A Distributed Methodology for Imbalanced Classification Problems. In: 2012 11th International Symposium on Parallel and Distributed Computing (ISPDC); 2012. pp. 164–71.
100. Hammond K, Varde AS. Cloud based predictive analytics: text classification, recommender systems and decision support. In: 2013 IEEE 13th International Conference on Data Mining Workshops; Dallas, TX, 2013, pp. 607–12.
101. Esteves RM, Pais R, Rong C. K-means Clustering in the Cloud—A Mahout Test. In: 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications; 2011. pp. 514–9.
102. Metz C. Mahout, There It Is! Open Source Algorithms Remake Overstock.com. Wired Magazine. 2012. http://www.wired.com/2012/12/mahout/. Accessed 18 Dec 2014.
103. Jack K. Mahout becomes a researcher: Large Scale Recommendations at Mendeley. In: Big Data Week, Hadoop User Group UK; 2012.
104. Sumbaly R, Kreps J, Shah S. The big data ecosystem at LinkedIn. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13); 2013. pp. 1125–34.
105. Ingersoll G. Apache Mahout: Scalable machine learning for everyone. IBM Corporation; 2011.
106. Sparks ER, Talwalkar A, Smith V, Kottalam J, Pan X, Gonzalez J, Franklin MJ, Jordan MI, Kraska T. MLI: An API for Distributed Machine Learning. In: 2013 IEEE 13th International Conference on Data Mining; 2013. pp. 1187–92.
107. Zhao H. High Performance Machine Learning through Codesign and Rooflining. Dissertation, University of California at Berkeley; 2014.
108. Peng H, Liang D, Choi C. Evaluating Parallel Logistic Regression Models. In: 2013 IEEE International Conference on Big Data; 2013. pp. 119–26.
109. Rennie JDM, Shih L, Teevan J, Karger DR. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003); 2003.

110. Ingersoll G. Introducing Apache Mahout: Scalable, commerical-friendly machine learning for building intelligent applications, IBM Corporation; 2009.
111. Owen S, Anil R, Dunning T, Friedman E. Mahout in Action. Shelter Island, NY;2011.
112. Wang Y, Wei J, Srivatsa M, Duan Y, Du W. IntegrityMR: Integrity assurance framework for big data analytics and management applications. In: 2013 IEEE International Conference on Big Data; 2013. pp. 33–40.
113. Verma A, Cherkasova L, Campbell RH. Play It Again, SimMR! In: 2011 IEEE International Conference on Cluster Computing; 2011. pp. 253–61.
114. Janeja VP, Azari A, Namayanja JM, Heilig B. B-dIDS: Mining Anomalies in a Big-distributed Intrusion Detection System. In: 2014 IEEE International Conference on Big Data; 2014. pp 32–4.
115. Singh K, Guntuku SC, Thakur A, Hota C. Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests. Inf Sci. 2014;278:488–97.
116. Racette MP, Smith CT, Cunningham MP, Heekin TA, Lemley JP, Mathieu RS. Improving situational awareness for humanitarian logistics through predictive modeling. In: Systems and Information Engineering Design Symposium (SIEDS); 2014. pp. 334–9.
117. Ko KD, El-Ghazawi T, Kim D, Morizono H. Predicting the severity of motor neuron disease progression using electronic health record data with a cloud computing Big Data approach. In: 2014 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology; 2014.
118. Li L, Bagheri S, Goote H, Hasan A, Hazard G. Risk Adjustment of Patient Expenditures: A Big Data Analytics Approach. In: 2013 IEEE International Conference on Big Data; 2013. pp. 12–4.
119. Zolfaghar K, Meadem N, Teredesai A, Roy SB, Chin S, Muckian B. Big data solutions for predicting risk-of-readmission for congestive heart failure patients. In: 2013 IEEE International Conference on Big Data; 2013. pp. 64–71.
120. Mylaraswamy D, Xu B, Dietrich P, Murugan A. Case Studies: Big Data Analytics for System Health Monitoring. In: 2014 International Conference on Artificial Intelligence (ICAI'14); 2014.
121. Esteves RM Rong C. Using Mahout for Clustering Wikipedia's Latest Articles: A Comparison between K-means and Fuzzy C-means in the Cloud. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science; 2011. pp. 565–9.
122. Filimon D. Clustering of Real-time Data at Scale. In: Berlin Buzzwords; 2013.
123. Gao F, Abd-Almageed W, Hefeeda M. Distributed approximate spectral clustering for large-scale datasets. In: Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing (HPDC'12); 2012. pp. 223–34.
124. Hussein T, Linder T, Gaulke W, Ziegler J. Hybreed: a software framework for developing context-aware hybrid recommender systems. User Model User-Adap Inter. 2014;24(1–2):121–74.
125. Yu H, Hsieh C, Si S, Dhillon IS. Parallel matrix factorization for recommender systems. Knowl Inf Syst. 2013;41(3):793–819.
126. Said A, Bellogín A. Comparative Recommender System Evaluation: Benchmarking Recommendation Frameworks. In: Proceedings of the 8th ACM Conference on Recommender systems (RecSys'14); 2014. pp. 129–36.
127. Zheng J, Dagnino A. An initial study of predictive machine learning analytics on large volumes of historical data for power system applications. In: 2014 IEEE International Conference on Big Data; 2014. pp. 952–59.
128. Katsipoulakis NR, Tian Y, Reinwald B, Pirahesh H. A Generic Solution to Integrate SQL and Analytics for Big Data. In: 18th International Conference on Extending Database Technology (EDBT); 2015. pp. 671–6.
129. Alber M. Big Data and Machine Learning: A Case Study with Bump Boost. Thesis, Free University of Berlin; 2014.
130. Lin CY, Tsai CH, Lee CP, Lin CJ. Large-scale logistic regression and linear support vector machines using spark. In: 2014 IEEE International Conference on Big Data; 2014. pp. 519–28.
131. Zhang C. DimmWitted: A Study of Main-Memory Statistical Analytics. 2014. arXiv Preprint,. arXiv:1403.7550.
132. Koutsoumpakis G. Spark-based Application for Abnormal Log Detection. Thesis, Uppsala University; 2014.
133. Powered By Spark. https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark. Accessed 15 Dec 2014.
134. Talwalkar A, Kraska T, Griffith R, Duchi J, Gonzalez J, Britz D, Pan X, Smith V, Sparks E, Wibisono A, Franklin MJ, Jordan MI. MLbase: A Distributed Machine Learning Wrapper. In: NIPS Big Learning Workshop; 2012.
135. Kraska T, Talwalkar A, Duchi J, Griffith R, Franklin MJ, Jordan M. MLbase: A Distributed Machine-learning System. In: 6th Biennial Conference on Innovative Data Systems Research; 2013.
136. Pan X, Sparks ER, Wibisono A. MLbase: Distributed Machine Learning Made Easy. University of California Berkeley Technical Report; 2013.
137. Sparks ER, Talwalkar A, Franklin MJ, Jordan MI, Kraska T. TuPAQ: an efficient planner for large-scale predictive analytic queries. 2015. **(arXiv Preprint arXiv:1502.00068)**.
138. Sparks E. Scalable Automated Model Search. University of California at Berkeley Technical Report UCB/EECS-2014-122; 2014.
139. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015;2(1):1–21.
140. Deeplearning4j. http://www.skymind.io/deeplearning4j/.
141. KNIME. http://www.knime.org/.
142. RapidMiner. https://rapidminer.com/.
143. H2O (2015) Algorithms Roadmap.
144. Kejela G, Esteves RM, Rong C. Predictive Analytics of Sensor Data Using Distributed Machine Learning Techniques. In: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science; 2014. pp. 626–31.
145. Morales GDF, Bifet A. SAMOA: Scalable Advanced Massive Online Analysis. J Mach Learn Res. 2015;16:149–53.
146. Bifet A, Morales GDF. Big Data Stream Learning with SAMOA. In: 2014 IEEE International Conference on Data Mining Workshop (ICDMW); 2014. pp. 1199–202.
147. Severien AL. Scalable Distributed Real-Time Clustering for Big Data Streams. Thesis, Polytechnic University of Catalonia; 2013.

148. Romsaiyud W. Automatic Extraction of Topics on Big Data Streams through Scalable Advanced Analysis. In: 2014 International Computer Science and Engineering Conference (ICSEC); 2014. pp. 255–60.
149. Riondato M, DeBrabant JA, Fonseca R, Upfal E. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12); 2012. pp. 85–94.
150. SAMOA-MOA. https://github.com/samoa-moa/samoa-moa.
151. Kourtellis N, Morales GDF, Bonchi F. Scalable Online Betweenness Centrality in Evolving Graphs; 2015. arXiv Preprint:1401.6981.
152. Rahnama AHA. Real-time Sentiment Analysis of Twitter Public Stream. Thesis, University of Jyväskylä; 2015.
153. Rahnama AHA, Sentinel. http://ambodi.github.io/sentinel/.
154. Qi B, Ma G, Shi Z, Wang W. Efficiently Finding Top-K Items from Evolving Distributed Data Streams. In: 2014 10th International Conference on Semantics, Knowledge and Grids (SKG); 2014.
155. Di Mauro M, Di Sarno C. A framework for Internet data real-time processing: A machine-learning approach. In: 2014 International Carnahan Conference on Security Technology (ICCST); 2014.
156. Distributed Weka. http://www.cs.waikato.ac.nz/ml/weka/bigdata.html.
157. Oryx. https://github.com/cloudera/oryx.
158. Oryx 2. https://github.com/OryxProject/oryx.
159. Vowpal Wabbit. https://github.com/JohnLangford/vowpal_wabbit.
160. Van Hulse J, Khoshgoftaar T. Knowledge discovery from imbalanced and noisy data. Data Knowl Eng. 2009;68(12):1513–42.
161. Khoshgoftaar TM, Hulse JV. Imputation techniques for multivariate missingness in software measurement data. Software Quality J. 16(4):563–600; 2008. [Online]. http://dx.doi.org/10.1007/s11219-008-9054-7.
162. Khoshgoftaar TM, Van Hulse J, Napolitano A. Comparing boosting and bagging techniques with noisy and imbalanced data. Syst Man Cybern Part A Syst Hum IEEE Trans. 2011;41(3):552–68.
163. Van Hulse J, Khoshgoftaar TM, Napolitano A, Wald R. Feature selection with high-dimensional imbalanced data. In: IEEE International Conference on Data Mining Workshops (ICDMW'09); 2009. pp. 507–14.
164. Van Hulse J, Khoshgoftaar TM, Napolitano A. Experimental perspectives on learning from imbalanced data. In: Proceedings of the 24th International Conference on Machine Learning; 2007. pp. 935–42.
165. Hogan JM, Peut T. Large Scale Read Classification for Next Generation Sequencing. Procedia Comput Sci. 2014;29:2003–12.
166. Sun K, Miao W, Zhang X, Rao R. An Improvement to Feature Selection of Random Forests on Spark. In: 2014 IEEE 17th International Conference on Computational Science and Engineering (CSE); 2014. pp. 774–9.
167. Kandel S, Paepcke A, Hellerstein JM, Heer J. Enterprise data analysis and visualization: an interview study. IEEE Trans Visual Comput Graphics. 2012;18(12):2917–26.
168. Kelley I, Blumenstock J. Computational Challenges in the Analysis of Large, Sparse, Spatiotemporal Data. In: Proceedings of the sixth international workshop on Data intensive distributed computing; 2014. pp. 41–5.
169. Ganz F, Puschmann D, Barnaghi P, Carrez F. A practical evaluation of information processing and abstraction techniques for the internet of things. IEEE Internet Things J; 2015 **(preprint).**