

Big Data Analytics with Machine Learning Tools

T.P. Fowdur, Y. Beeharry, V. Hurbungs, V. Bassoo
and V. Ramnarain-Seetohul

Abstract Big data analytics is the current, trending hot-topic in the research community. Several tools and techniques for handling and analysing structured and unstructured data are emerging very rapidly. However, most of the tools require high expert knowledge for understanding their concepts and utilizing them. This chapter presents an in-depth overview of the various corporate and open-source tools currently being used for analysing and learning from big data. An overview of the most common platforms such as IBM, HPE, SAPRANA, Microsoft Azure and Oracle is first given. Additionally, emphasis has been laid on two open-source tools: H2O and Spark MLlib. H2O was developed by H2O.ai, a company launched in 2011 and MLIB is an open source API that is part of the Apache Software Foundation. Different classification algorithms have been applied to Mobile-Health related data using both H2O and Spark MLlib. Random Forest, Naïve Bayes, and Deep Learning algorithms have been used on the data in H2O. Random Forest, Decision Trees, and Multinomial Logistic Regression Classification algorithms have been used with the data in Spark MLlib. The illustrations demonstrate the

T.P. Fowdur (✉)

Department of Electrical and Electronic Engineering, Faculty of Engineering,
University of Mauritius, Réduit, Moka, Mauritius
e-mail: p.fowdur@uom.ac.mu

Y. Beeharry

Faculty of Information, Communication and Digital Technologies,
University of Mauritius, Réduit, Moka, Mauritius
e-mail: y.beeharry@uom.ac.mu

V. Hurbungs

Department Software and Information Systems, Faculty of Information,
Communication and Digital Technologies, University of Mauritius, Réduit, Mauritius
e-mail: v.hurbungs@uom.ac.mu

V. Bassoo · V. Ramnarain-Seetohul

Department of Information Communication Technology, Faculty of Information,
Communication and Digital Technologies, University of Mauritius, Réduit, Mauritius
e-mail: v.bassoo@uom.ac.mu

V. Ramnarain-Seetohul

e-mail: v.seetohul@uom.ac.mu

© Springer International Publishing AG 2018

N. Dey et al. (eds.), *Internet of Things and Big Data Analytics Toward
Next-Generation Intelligence*, Studies in Big Data 30,
DOI 10.1007/978-3-319-60435-0_3

flows for developing, training, and testing mathematical models in order to obtain insights from M-Health data using open source tools developed for handling big data.

Keywords Open-source tools · H2O · Spark MLlib

1 Introduction

There are several conventional corporate tools which can be used to process data efficiently. This brings about useful insights for every department ranging from profit forecasting in sales to inventory management. The conventional architecture consists of a data warehouse where the data typically resides and Standard Query Language (SQL)-based Business Intelligence (BI) tools which execute the tasks of processing and analysing this same data [1]. In order to perform data analysis in a more advanced manner, companies have used dedicated servers to which the data was moved to perform data and text mining, statistical analysis and predictive analytics. The criticality of business analysis and profit making had brought about the wide usage [2] of tools like R [3] and Weka [4] in the deployment of machine learning algorithms and models. However, the evolution and growth of traditional data into what is commonly termed as Big Data nowadays has rendered the conventional tools unusable due to the major challenges resulting from the features of Big Data.

Volume, variety and velocity, which are the three main features of big data, have caused it to surpass the capabilities of its ingestion, storage, analysis and processing by conventional systems. Big Data means very large data sets, with both structured and unstructured data, that come at high speed from different sources. In addition, not all incoming data are valuable; some of it may not be of good quality carrying a certain level of uncertainty. The analysis conducted by the International Data Corporation's annual Digital Universe [5] has shown that the expected growth in data volume will be around 44 zettabytes (4.4×10^{22} bytes) by the year 2020 which is approximately ten times larger than it was in 2013. This growth can be explained by the increasing adoption of social media (e.g. Facebook, Twitter), Internet of Things (IoT) sensors, click streams from websites, video camera recordings and smartphones communications.

Handling huge amounts of data requires equipment and skills. Many organizations are already in possession of these equipment and expertise for managing and analysing structured data only. However, the fast flows and increasing volumes of data leave them with the inability to "mine" and extract intelligent actions in a timely manner. In addition to the fast growth of the volume of data which cannot be handled for traditional analytics, the variety of data types and speed at which the data arrives requires novel solutions for data processing and analytics [2]. Compared to conventional structured data of organisations, Big Data does not always fit into neat tables of rows and columns. New data types pertaining to

structured and unstructured data have come into existence and which are processed to yield business insights. The very common toolkits for machine learning such as Weka [4] or R [3] have not been conceived for handling such workloads. Despite having distributed applications of some available algorithms, Weka does not match the level of the initial tools developed for handling data at the terabyte scale. Therefore, traditional analytics tools are not well adapted to capture the full value of large data sets like Big Data.

Machine learning allows computers to learn without being explicitly programmed; the programmes have the ability to change when exposed to different sets of data. Machine learning techniques are therefore more appropriate to capture hidden insights from Big Data. The whole process is data-driven and executes at the machine layer by applying complex mathematical calculations to Big Data. Machine learning requires large data sets for learning and discovering patterns in data. Commonly used machine learning algorithms are: Linear Regression, Logistic Regression, Decision Tree, Naive Bayes and Random Forest. There are several tools that have been developed for working with Big data for example JasperSoft, Pentaho, Tableau and Karmasphere. These tools alleviate the problem of scalability by providing solutions for parallel storage and processing [6].

Businesses are increasingly adopting Big Data technologies to remain competitive and therefore, the demand for data management is on the rise given the advantages associated with analytics. Emerging big data applications areas are Healthcare, Manufacturing, Transportation, Traffic management, Media and Sports among others. Extensive research and development is also being carried out in the field of Big Data, analytics and machine learning. In [7], the authors highlight an expandable mining technique using the *k*-nearest neighbor (*K*-NN) classification approach to demonstrate training data with reduced data sets. This reduced data set ensures faster data classification and storage management. In [8], a methodology to clean raw data using the Fishers Discrimination Criterion (FDC) is proposed. The objective was to ease the data flow across a social network such as Facebook to enhance user satisfaction using machine learning techniques. Excessive data sets may result in the field of biological science (e.g. DNA information) and this adds to the difficulty of extracting meaningful information. A system to reduce similar information from large data sets was therefore proposed; the study analysed processing techniques such as NeuralSF and Bloom Filter applied to biological data [9]. Another biological study attempts to train a neural network using nine parameters to categorise breast cancer as cancerous or non-cancerous [10]. This study applies machine learning techniques on breast cancer dataset to accurately classify this type of cancer with an accuracy of 99.27%. Other example of neural network training is the performance comparison of NN-GA with NN, SVM and Random Forest classifiers to predict the categorisation of a pixel image [11]. In the field of computing and intelligence, related works include cryptographic key management [12] and intelligent activities of daily living (ADLs) assistance systems [13].

Big Data undoubtedly brings many benefits to businesses. However, there are many challenges associated with Big Data and related concepts. First of all, Big

Data changes the whole concept of data storage and manipulation. Businesses must now handle a constant inflow of data and this involves modifying the data management strategies of organisations. Secondly, privacy issues are associated with Big Data since the information is being captured from different sources. In the process, sensitive data may be processed without the knowledge of users. Thirdly, businesses have to invest in new technologies since the whole software environment is changing with Big Data. The latter implies super computing power for intensive data storage, processing capabilities and new software or hardware to handle analytics and machine learning. Overall, Big Data brings forward a new approach to handle organisational data and therefore, businesses must properly balance the challenging aspects of Big Data.

1.1 *Big Data Analytic Tools*

An overview of some existing corporate-level Big Data platforms is given in the following sub-sections.

1.1.1 **Big Data Analytic Tools**

IBM solution is currently helping businesses to achieve tangible benefits by analyzing all of their data available. IBM enables rapid return on investment across several areas such as Financial Services, Transportation, Healthcare/Life Sciences, Telecommunications, Energy and Utilities, Digital Media, Retail and Law Enforcement [14]. A few challenges and benefits of IBM in three areas are summarised in Table 1.

IBM gives enterprises the ability to manage all their data with a single integrated platform, multiple data engines and a consistent set of tools & utilities to operate on Big Data. IBM delivers its Big Data platform through a “Lego block” approach [15] and the platform has four core capabilities [16]:

Table 1 Big data challenges and benefits [14]

Area	Big data challenge	Benefit
Healthcare	Analyzing streaming patient data	Reduction in patients’ mortality by 20%
Telecommunication	Analyzing networking and call data	Reduction in processing time by 92%
Utilities	Analyzing petabytes of untapped data	99% improved accuracy in placing power generation resources

1. **Analytics using Hadoop:** It involves the analysis and processing of data in any format using clusters of commodity hardware.
2. **Stream Computing:** It involves real-time analysis of huge volumes of data on the fly with response times of the order of millisecond.
3. **Data Warehousing:** It involves advanced analytics on data stored in databases and the delivery of proper actionable insights.
4. **Information Integration and Governance:** It involves the processes of understanding, transforming, cleansing, delivering and governing reliable information to critical business initiatives [16].

Supporting Platform Services include Visualization & Discovery, Application Development, Systems Management and Accelerators [16]:

- **Visualization & Discovery:** It involves the exploration of complex and huge sets of data.
- **Application Development:** It involves the development of applications pertaining to big data.
- **Systems Management:** It involves the securing and optimizing of the performance of systems for big data.
- **Accelerators:** It involves the analytic applications that accelerate the development and implementation of big data solutions (Fig. 1).

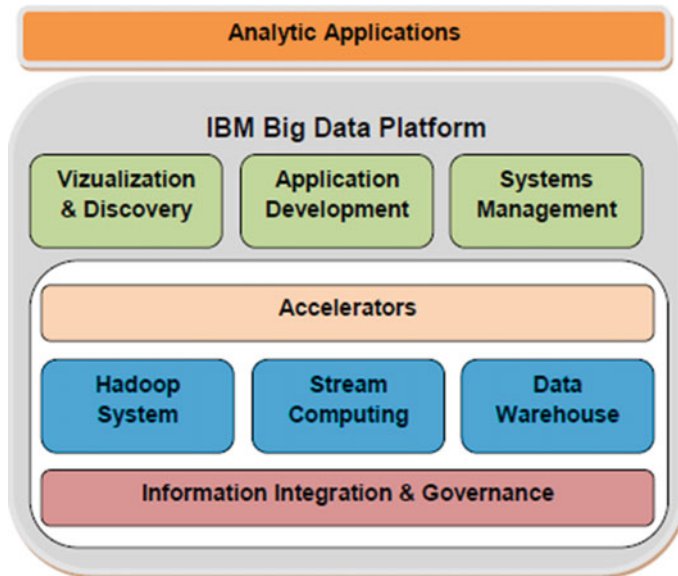


Fig. 1 IBM big data platform [17]

Table 2 Big data software, solutions and services [19]

Big data software	Solutions	Services
SQL analytics	Vertical advanced analytics	Platform for advanced analytics to obtain valuable insights from big data
Data analytics	IDOL	Unstructured data analytics across multiple languages
Big data verticals	Big Data industry solutions	Enterprise related issues
Big data cloud	Haven on demand	Big data platform to analyse data and create applications/services on the cloud

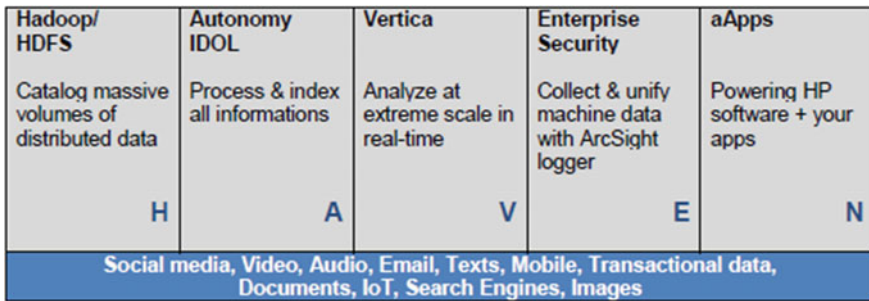


Fig. 2 Haven big data platform [20]

1.1.2 Hewlett-Packard Enterprise (HPE) Big Data Platform

HPE big data is based on an open architecture which improves existing business intelligence and analytics assets. The platform consists of scalable and flexible models, including on-premises, as-a-service, hosted, and private cloud solutions [18]. HPE offers comprehensive solutions to help an organization bridge to tomorrow through a data-driven foundation. HPE big data Software, Solutions and Services are highlighted in Table 2.

HP Haven OnDemand enables exploration and analysis of massive volumes of data including unstructured information such as text, audio, and video. Figure 2 highlights the components of the Haven big data platform which integrates analytics engines, data connectors and community applications. The main objective of this platform is further analysis and visualisation of data.

1.1.3 SAP HANA Platform

SAP HANA Vora [21] is a distributed computing and an in-memory solution which can aid organisations process big data to obtain insights profitable to the business

growth. Organisations use it for the quick and easy running of interactive analytics on both Hadoop and enterprise data. SAP HANA Vora single solution [21] for computation of all data includes Science, Predictive Analytics, Business Intelligence and Vizualisation Apps. Capabilities of the SAP HANA Platform can be categorised under the following groups [22]:

- **Application Services:** Making use of open standards for development such as: Java Script, HTML5, SQL, JSON, and others, to build applications on the web.
- **Database Services:** Performing advanced analytics and transactions on different types of data stored in-memory.
- **Integration Services:** Managing data collected from various sources and developing strategies to integrate all of them together.

According to [23], the SAP HANA Platform is most suitable to perform analytics in real-time, and develop and deploy applications which can process real-time information. The SAP HANA in-memory database is different from other database engines currently available on the market. This type of database provides high speed interactivity and fast response time for real-time applications such as Sales & Operational Planning, Cash Forecasting, Manufacturing scheduling optimization, point-of-sale data and social media data [23] (Fig. 3).

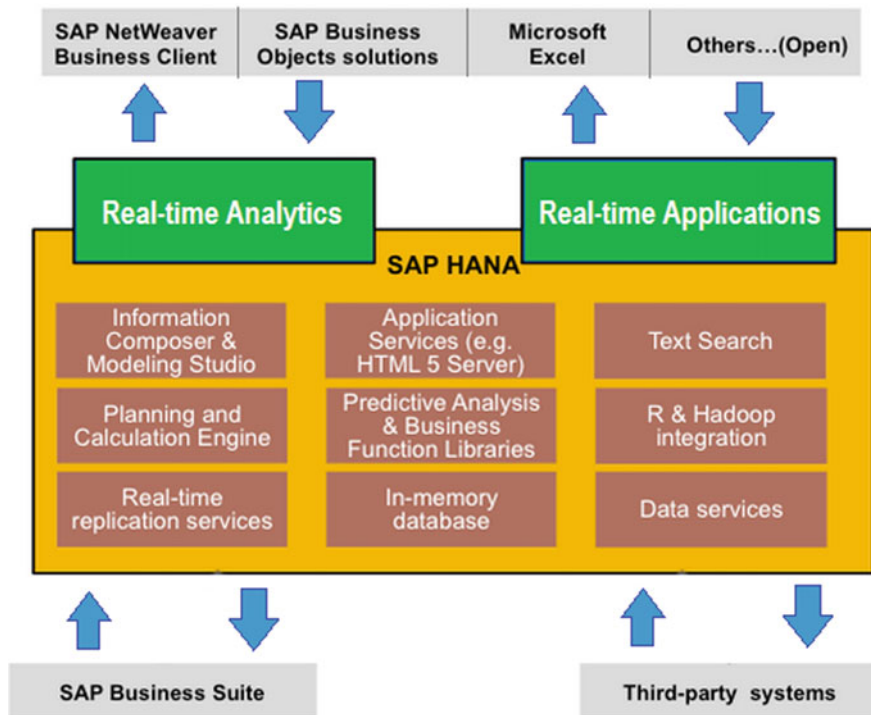


Fig. 3 SAP HANA platform [23]

1.1.4 Microsoft Azure

Microsoft Azure is an enterprise-level cloud platform which is flexible and open and allows for the fast development, deployment, and management of application across Microsoft data centers [24]. Azure complements existing IT environments by providing services in the cloud. Azure Big data and analytics services and products consist of [25]:

- **HDInsight:** R Server Hadoop, HBase, Storm, and Spark clusters.
- **Data Lake Analytics:** A service for performing analytics in a distributed environment which eases the processing of big data.
- **Machine Learning:** Very powerful tools on the cloud platform which allow predictive maintenance.

The important constituents of the Azure Services Platform are as follows [26]:

- **Windows Azure:** Networking and computation, storage with scalability, management and hosting services.
- **Microsoft SQL Services:** Services for storage and reporting using databases.
- **Microsoft.NET Services:** Services for implementing .NET Frameworks.
- **Live Services:** Services for distributing, maintaining and synchronizing files, photos, documents, and other information.
- **Microsoft Dynamics CRM Services and Microsoft SharePoint Services:** Fast solution development for Business collaboration in the cloud (Fig. 4).

World-class companies are doing amazing things with Azure. In its promotional campaign, Azure cloud is used by Heineken to target 10.5 million consumers [27]. Rolls-Royce and Microsoft collaborate to create new digital capabilities by not focussing on the infrastructure management but rather on the delivery of real value to customers [27]. Over 450 million fans of the Real Madrid C.F. worldwide have been able to access the stadium with the Microsoft Cloud [27]. NBC News chooses Microsoft Azure to Host Push Notifications to Mobile Applications [27]. 3M



Fig. 4 Azure services platform [26]

accelerates mobile application development and benefits from real-time insight with Azure cloud [27].

1.1.5 Oracle Big Data

Oracle provides the power of both Spark and Hadoop ready to be incorporated with the existing data of enterprises which were previously using Oracle applications and Database. The service provided is highly performant, secure and fully automated. Oracle Big Data offers an integrated set of products to organise and analyse various data sources to find new insights and take advantage of hidden relationships. Oracle Big Data features include [28, 29]:

- **Provisioning and Elasticity:** On demand Hadoop and Spark clusters
- **Comprehensive Software:** Big Data Connectors, Big Data Spatial and Graph, Advanced Analytics
- **Management:** Security, Automated life-cycle
- **Integration:** Object Store Integration, Big Data SQL (Fig. 5).

According to [31], Oracle helps Dubai Smart Government Establishment (DSG) in providing smart government services in the region by proposing new strategies and overseeing processes at the level of government entities. Using Oracle Big Data, the public health services of England has developed a fraud and error detection solution by analysing hidden patterns in physician prescription records [31].

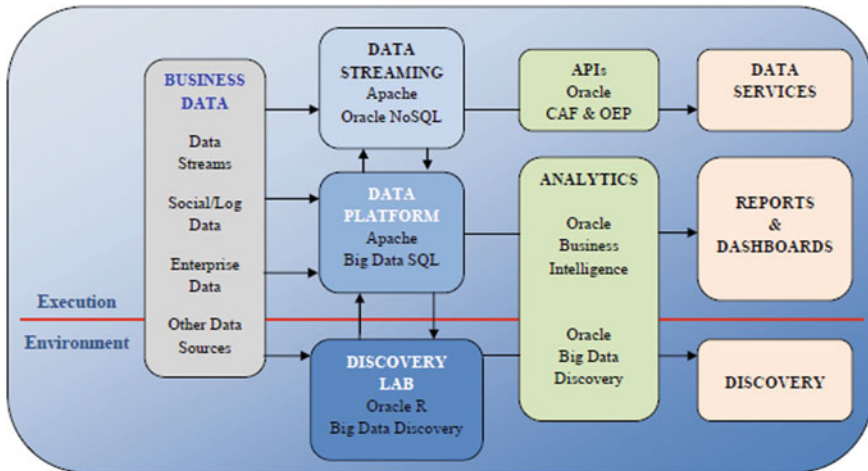


Fig. 5 Oracle solution for big data [30]

1.1.6 Other Big Data Platforms

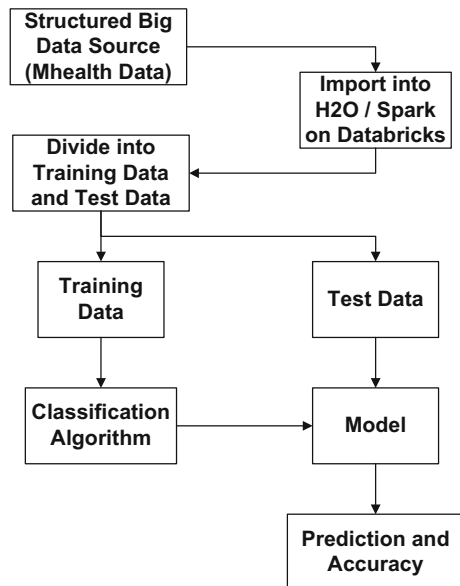
Other Big Data platforms and Big Data Analytics Software include Jaspersoft BI Suite, Talend open studio, Dell Big Data analytics, Pentaho Business analytics, Redhat Big Data, Splunk, Tableau, Karmasphere Studio and Analyst, Skytree Server.

1.2 Machine Learning Tools for Big Data

Several open-source tools have emerged in recent years for handling massive amounts of data. In addition to managing the data, obtaining insights from the latter is of utmost importance. As such, machine learning techniques add value in all the fields having huge amounts of data ready to be processed. In this chapter two of the open source tools namely H2O [32] and MLIB [33], which facilitate the collaboration of data and computer scientists, will be described in details.

The framework for big data analytics with machine learning is shown in Fig. 6. The structured data from the big data source is imported into H2O and Spark on Databricks. The Data frame obtained is then divided into a training set and a test set. The training set is used to train the machine learning model. Based on the algorithm used, the model is tested using the test set by predicted the labels from the features. The corresponding accuracy is finally computed.

Fig. 6 Framework for big data analytics with machine learning



1.2.1 H2O

H2O is an open-source software used with the trending big-data analytics [32]. The start-up company H2O.ai launched in 2011 in Silicon Valley [34] has brought H2O into existence. As part of pattern mining in data, several models can be fitted by H2O. It provides a fast open source software that supports parallel in-memory machine learning, predictive analytics and deep learning. It is based on pure Java and Apache v2 Open Source. It provides simple deployment with a single jar and automatic cloud discovery. H2O allows data to be used without sampling and provides reliable predictions quicker. The main aim of the H2O project is to bring forth a cloud computing interface, analytics, giving the necessary resources for data analytics to its users. H2O is open-source and has a free license. The profits of the company originate mainly from the service provided to customers and tailored extensions. According to VentureBeat in November 2014, H2O's clients included Nielsen, Cisco, eBay, and PayPal [35–37].

The chief executive of H2O is Sri Satish Ambati had assisted the setting up of a firm for big-data called Platfora. The software for the Apache Hadoop distributed file system is implemented by Platfora [38]. However, the R programming language did not provide satisfactory performance on large data-sets. Ambati therefore began developing the H2O software. The pioneer of the S language at Bell Labs, John Chambers who is also part of the R's leading development team, supported the work [37, 39, 40].

Oxdata was co-founded by Ambati and Cliff Click. Cliff held the post of the Chief Technical Officer (CTO) of H2O and had a significant input in H2O's products. Click gave a helping hand in writing the HotSpot Server Compiler and even collaborated with Azul Systems for the implementation of a big-data Java virtual machine (JVM) [41]. In February 2016, Click moved away from H2O [42]. The author of Grammar of Graphics, Leland Wilkinson, serves as Chief Scientist and provides visualization leadership [32].

Three professors at the Stanford University [43] have been listed as Mathematical Scientists by H2O's Scientific Advisory Council. One of them is Professor Robert Tibshirani, who partners with Bradley Efron on bootstrapping [44]. He also specialises on generalized additive models as well as statistical learning theory [45, 46]. The second one is Professor Stephen P. Boyd who specialises in convex minimization and applications in statistics and electrical engineering [47]. The third is Professor Trevor Hastie who is a collaborator of John Chambers on S [40], and also an expert on generalized additive models and statistical learning theory [45, 46, 48].

H2O Architecture and Features

The main features of the H2O software are given in Table 3 [34].

Table 3 Features of the H2O software [34]

H2O components	
Component	Description
H2O cluster	It supports clusters with multiple nodes. The system is modeled by taking the concept of shared memory into consideration. This is done since all the computations are performed in memory. Each node sees only a few rows from the dataset and the cluster size can grow without any limit
Distributed key value store	A key is employed to reference the objects such as models, data frames, and results in the H2O cluster
H2O frame	Distributed data frames are supported. Distributed arrays are used across all the nodes for the columns of the dataset. The whole dataset is visible to each node by using the HDFS
Distributed key-value store	
Feature	Description
Peer-to-peer	The H2O Key-Value Store is a typical peer-to-peer scattered hash table. There is neither a central key dictionary nor a name-node
Pseudo-random hash	Each key is associated to a home-node. The homes are selected using a pseudo-random algorithms for each key. This technique enables load-balancing
Key's home node	A key's "home" controls the starting and stopping of writes performed. Caching of Keys can be performed anywhere. A write is complete only upon reaching "home"
Data in H2O	
Characteristic	Description
Highly compressed	Data is compressed up to 4 times more as compared to gzip and is read from NFS and HDFS
Speed	The speed is dependent on the memory rather than the CPU. If linear data is performed, the speed is similar to that of Fortran or C. The speed is a ratio of the data volume to the memory bandwidth. It is approximately 50 GB/s and depends on the hardware
Data shape	Its operation is fast for a Table width <1 k works for a width <10 k but is slow for <100 k. The Table length is limited only by memory
Processing data in H2O	
Operation	Description
Map reduce	It provides a convenient way for developing parallel codes and it supports a relatively fast and efficient method. It also supports distributed fork/join and parallel map and inside every node, the conventional fork/join is supported
Group by	It supports groups of the order of one million on billions of rows, and executes Map/Reduce operations on the elements of the group
Others	All the simple data frame operations of H2O have been integrated in R and Python. Within the algorithms, transactions such as imputation and one-hot encoding of categories are executed

(continued)

Table 3 (continued)

Communication in H2O	
Feature	Description
Network communication	The disparate processes or machine memory spaces need connectivity to JVMs. The network speed of the connection can be fast or slow and there can even be dropped packets & sockets. Even TCP can fail in some cases and retries might be required
Reliable RPC	H2O builds a dependable RPC which restarts dropped connections at the RPC level. Recovery of the cluster can be achieved by pulling cables from a running cluster and re-plugging them
Optimizations	Compression of data are achieved in several ways: to save network bandwidth, UDP packets are used for transmitting short messages and TCP provides congestion control for larger messages

The functionalities for Deep Learning using H2O are as follows [35]:

- Undertaking of tasks pertaining to classification and regression by making use of supervised training models.
- Operations in Java using refined Map/Reduce model and which increase the efficiency in terms of speed and memory.
- Making usage of cluster with multiple nodes or a single node running distributed and Multi-threaded computations simultaneously.
- Availability of the options to set annealing, momentum, and learning rate. In addition, in order to achieve convergence as fast as possible, a completely adaptive and automatic learning rate per-neuron is also available.
- Availability of model averaging, and L1 and L2 Loss function and regularization to avoid over fitting of the model.
- Possibility of selecting the model from a user-friendly web-based interface as well as using scripts in R equipped with the H2O package.
- Availability of check-points to help achieve lowered run times as well as tune the models.
- Availability of the automatic process of discarding missing values in the numerical and categorical datasets during both the pre-processing and post-processing stages.
- Availability of the automatic feature which handles the balance between computation and communication over the cluster to achieve the best performance. Additionally, there is the possibility of exporting the designed model in java code and use in production environments.
- Availability of additional features and parameters for adjustments in the model automatic encoders for handling unsupervised learning models and ability to detect anomalies.

A typical architecture of the H2O software stack is shown in Fig. 7 [49].

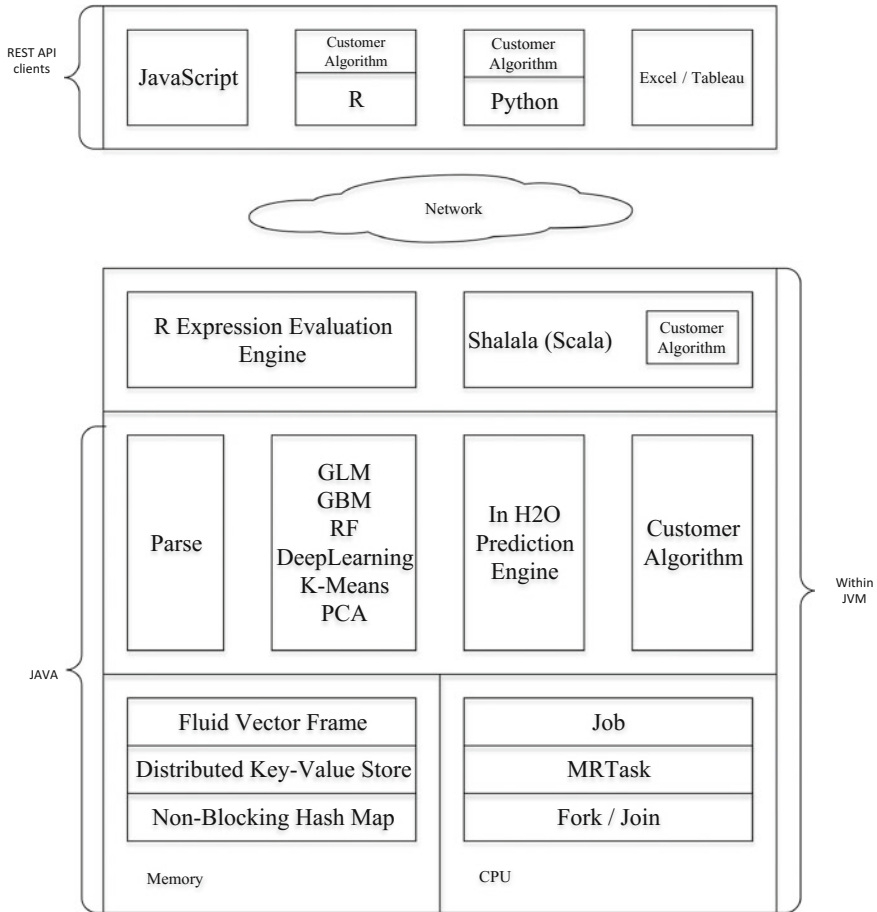


Fig. 7 H2O software architecture [49]

A socket connection is used by all the REST API clients for communication with H2O. JavaScript is the language used to code the embedded H2O Web UI which uses the standard REST API. The H2O R package “library(h2o)” can be used in R scripts. Instead of executing H2O using ‘apply’ or ‘ddply’, R functions can be written by the users. The way the system is at present, the REST API must be used directly in Python scripts. The commonly used Microsoft Excel is also equipped with an H2O worksheet. This enables importing large volumes of data into H2O and the running of algorithms such as GLM directly from Excel. Visualization in Tableau is also possible by pulling results directly from H2O.

One or more nodes combined make up an H2O cloud. The JVM process, which consists of a single node, is divided into three layers: core infrastructure, algorithms, and language.

The Shalala Scala and R layer comprise of a language layer consisting of engine for evaluation of expressions. The R REST client front-end is master to the evaluation layer of R. The Scala layer is different from the R layer in the sense that algorithms and native programs using H2O can be written directly. The algorithms layer is made up of all the default algorithms in H2O [49]. For example, datasets are imported using the parse algorithm. It also includes GLM and algorithms for predictive analytics and model evaluation. The bottom (core) layer manages Memory and CPU resources.

The elementary unit of data storage which the users are exposed to is an H2O Data Frame. The engineering term referring to the capability of adding, removing, and updating columns fluidly in a frame is “Fluid Vector”. It is the column-compressed store implementation.

The distributed Key-Value store are essentially atomic and distributed in-memory storage and spread across the cluster. The Non-blocking Hash Map is used in the Key-Value store implementation. CPU Management is based on three main parts. First, there are jobs which are large pieces of work having progress bars and which can be monitored using the web user interface. One example of a job is Model creation. Second there is the MapReduce Task (MRTask). This is an H2O in-memory MapReduce Task and not a Hadoop MapReduce task. Finally there is Fork/Join which is a modified version of the JSR166y lightweight task execution framework.

H2O can support billions of data rows in-memory even if the cluster size is relatively small. This is possible by the use of sophisticated in-memory compression techniques. The H2O platform has its own built-in Flow web interface so as to make analytic workflows become user-friendly to users who do not have engineering background. It also includes interfaces for Java, Python, R, Scala, JSON and Coffeescript/JavaScript. The H2O platform was built alongside and on top of both Hadoop and Spark Clusters. The deployment is typically done within minutes [35, 36, 43]. The software layers that are associated with the running of an R program to begin a GLM on H2O are shown in Fig. 8.

Figure 8 shows two different sections. The section on the left depicts the steps which run the R process and the section on the right depicts the steps which run in the H2O cloud. The TCP/IP network code lies above the two processes allowing them to communicate with each other. The arrows with solid lines demonstrate a request originating from R to H2O and the arrows with dashed lines demonstrate the response from H2O to R for the request. The different components of the R program are: R script, R package for R script, (RCurl, Rjson) packages, and R core runtime. Figure 9 shows the R program retrieving the resulting GLM model [49].

The H2OParsedData class consists of an S4 object which is used to represent an H2O data frame in R. The big data object in H2O is referenced by an @key slot in the S4 object. The H2OParsedData class is used by the package in R to perform summarization and merging operations. These operations are transmitted over to the H2O cloud by making use of an HTTP connection. The operation is performed on the H2O cloud platform and the result is returned as a reference which is in turn stored in another S4 object in R.

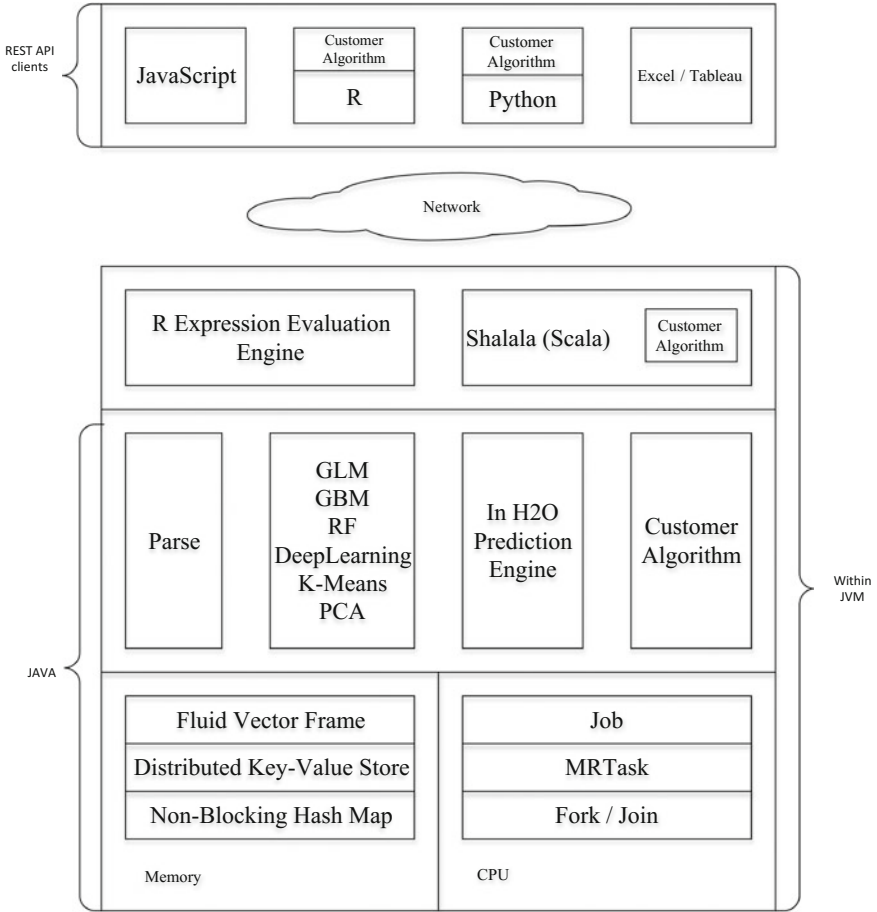


Fig. 8 R script starting H2O GLM

H2O Algorithms

Several common data analysis and machine learning algorithms are supported by H2O. These algorithms can be classified as per Table 4 [36, 49].

H2O Deployment and Application Example

H2O can be launched by opening a command prompt terminal, moving to the location of the jar file, and executing the command: java—jar h2o.jar as shown in Fig. 10.

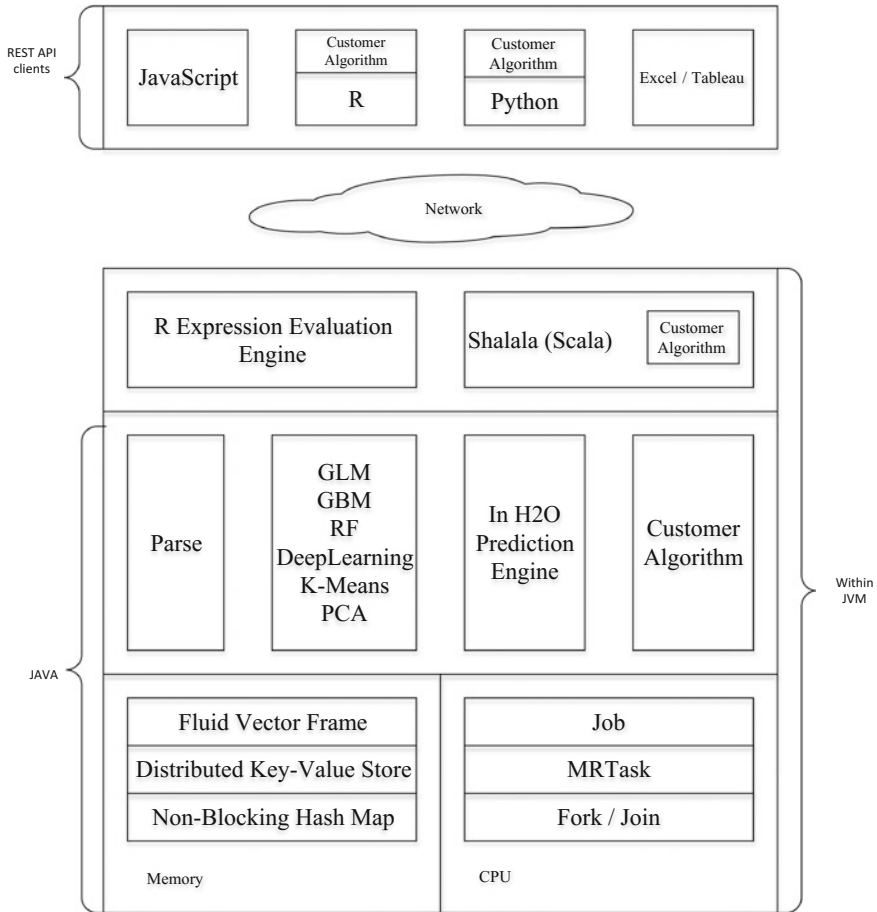


Fig. 9 R Script retrieving the resulting GLM model

To access the web interface, type-in the url: <https://localhost:54321/flow/index.html> in the web browser. The start-up page shows a list of functionalities supported in H2O. This can also be viewed by using the *assist* option as depicted in Fig. 11.

In this section, the Mobile Health (MHEALTH) dataset from [50–52] has been used to demonstrate classification machine learning algorithms with H2O.

The dataset consists of recordings of body motions for volunteers while performing different physical activities. The twelve activities recorded are:

- L1 Standing still
- L2 Sitting and Relaxing
- L3 Lying down
- L4 Walking
- L5 Climbing stairs

Table 4 Overview of H2O algorithms [49]

Category	Techniques
Regressions/Generalized Linear Modeling (GLM)	<ul style="list-style-type: none"> • GLMNet • Distributions: Gaussian, Binomial, Poisson, Gamma, Tweedie • Bayesian regression • Multinomial regression
Classifications	<ul style="list-style-type: none"> • Distributed random forest • Gradient boosting machine • Distributed trees • Naïve Bayes
Neural networks	<ul style="list-style-type: none"> • Multi-layer perceptron • Auto-encoder • Restricted Boltzmann machines
Solvers and optimization	<ul style="list-style-type: none"> • Generalized ADMM solver • BFGS (quasi newton method) • Ordinary least square solver • Stochastic gradient descent MCMC
Clustering	<ul style="list-style-type: none"> • K-means, • K-nearest neighbors • Locality sensitive hashing • Dimensionality reduction • Singular value decomposition
Recommendation	<ul style="list-style-type: none"> • Collaborative filtering • Alternating least squares
Time series	<ul style="list-style-type: none"> • ARIMA, ARMA modeling • Forecasting
Data munging	<ul style="list-style-type: none"> • Plyr • Integrated R-environment • Slice, Log transform • Anonymizing/Obfuscating (for personalized or confidential data)

- L6 Waist bends forward
- L7 Frontal elevation of arms
- L8 Knees bending (crouching)
- L9 Cycling
- L10 Jogging
- L11 Running
- L12 Jump front and back.

In addition to these known classes during which experimentation is performed, there is also a Null class with a value of '0' for data recorded when none of the above recordings are being performed. The feature values are mainly obtained from sensors: acceleration from chest sensor, electrocardiogram sensor, accelerometer sensor, gyro sensor, and magnetometer sensor. The accelerometer, gyro, and magnetometer sensors are placed on the arms and ankles of the subjects.

```
Command Prompt - java -jar h2o.jar
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\LabUser>cd C:\Users\LabUser\Desktop\H2O

C:\Users\LabUser\Desktop\H2O>java -jar h2o.jar
02-13 10:54:21.909 172.22.45.24:54321 7652 main INFO: ----- H2O starte
d
02-13 10:54:22.027 172.22.45.24:54321 7652 main INFO: Build git branch
: rel-tverberg
02-13 10:54:22.028 172.22.45.24:54321 7652 main INFO: Build git hash:
42269eb55b790f609e69c9f32897f7ee405f3b71
02-13 10:54:22.028 172.22.45.24:54321 7652 main INFO: Build git descri
be: jenkins-rel-tverberg-4
02-13 10:54:22.029 172.22.45.24:54321 7652 main INFO: Build project ve
rsion: 3.10.3.4 (latest version: 3.10.3.4)
02-13 10:54:22.029 172.22.45.24:54321 7652 main INFO: Build age: 9 day
s
02-13 10:54:22.029 172.22.45.24:54321 7652 main INFO: Built by: 'jenki
ns'
02-13 10:54:22.030 172.22.45.24:54321 7652 main INFO: Built on: '2017-
02-03 12:29:19'
02-13 10:54:22.030 172.22.45.24:54321 7652 main INFO: Processed H2O ar
guments: []
02-13 10:54:22.031 172.22.45.24:54321 7652 main INFO: Java availablePr
ocessors: 4
```

Fig. 10 Launching of H2O through the command prompt

assist

? Assistance

Routine	Description
importFiles	Import file(s) into H ₂ O
getFrames	Get a list of frames in H ₂ O
splitFrame	Split a frame into two or more frames
mergeFrames	Merge two frames into one
getModel	Get a list of models in H ₂ O
getGrids	Get a list of grid search results in H ₂ O
getPredictions	Get a list of predictions in H ₂ O
getJobs	Get a list of jobs running in H ₂ O
buildModel	Build a model
importModel	Import a saved model
predict	Make a prediction

Fig. 11 Assistance list for operations available in H2O

The common dataset to be used with the classification algorithms can be imported using the *importFiles* option in the list on the homepage of H2O as depicted in Fig. 12.

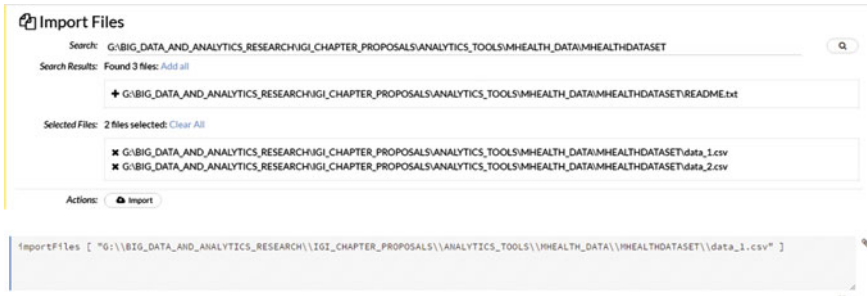


Fig. 12 Import data set on the cluster to be processed in H2O

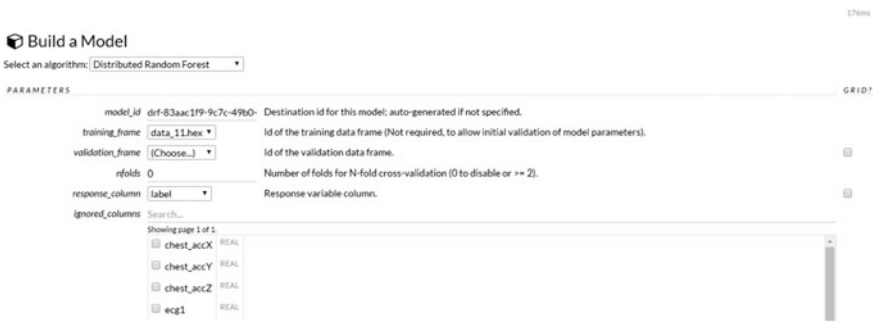


Fig. 13 Building a distributed random forest model

As can be observed from Fig. 12, two csv-files are loaded in H2O. The file ‘data1.csv’ is used for training of the different models and ‘data2.csv’ is used for testing.

Illustration 1: Random Forest Classification in H2O

The model is then built using the training data set as input. The Distributed Random Forest (DRF) is used with the “label” column selected as the response column. These are shown in Fig. 13. The equation being modelled is: $label \sim chestAccX + chestAccY + \dots$

The output metrics on the training model is shown in Fig. 14.

With the training model obtained, prediction can be performed on the test set data. A comparison can then be performed on the predicted and already known “label”. Figure 15 shows the step where the model is used to predict the “label” for the test set.

The predicted “label” can be merged with the test data set and compared with that already known. Figure 16 shows the confusion matrix for the test data set.

An analysis of the percentage difference between the predicted and already known “label” for the test data set can be performed as shown in Table 5.

```

▼ OUTPUT - TRAINING_METRICS
model drf-dfb6e02f-cd40-4383-88e2-09c055796131
model_checksum -3755657836809008128
frame data_1_NB.hex
frame_checksum 30919538483876304
description Metrics reported on Out-Of-Bag training samples
model_category Multinomial
scoring_time 1487413838996
predictions .
MSE 0.061304
RMSE 0.247597
nobs 982273
r2 0.994981
logloss 0.206972
mean_per_class_error 0.113143

```

Fig. 14 Output metrics for training model

predict

Predict

Name: prediction-92e35ec2-f8f

Model: drf-dfb6e02f-cd40-4383-88e2-09c055796131

Frame: data_2_NB.hex

Actions: Predict

predict model: "drf-dfb6e02f-cd40-4383-88e2-09c055796131", frame: "data_2_NB.hex", predictions_frame: "prediction-92e35ec2-f8ff-4caf-9f34-ce43ec5e8ae7"

Fig. 15 Prediction using training model on test data set

*PREDICTION - CM - CONFUSION MATRIX

	Null	clocking_store	cycling_framas	franta_cake_merits_ame	gagging_funs	front_out_baoi	hones_banding	lyng_dow	numing	setting_and_relaxing	steaming_still	stret_benaf	forward	walking	franta_cake_merits_ame	Error Rate
285041	328	136	0	754	144	38	2348	2421	0	0	0	37	243	0	0.1828	5,387 / 185,428
4139	2085	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1737	4,339 / 8,344
4559	0	2385	0	0	0	0	0	0	0	0	0	0	0	0	0.7428	4,339 / 8,344
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 0
2857	0	0	0	3936	1	0	0	0	0	0	0	0	0	0	0.4294	2,108 / 8,344
3742	0	0	0	0	332	0	0	0	0	0	0	0	0	0	0.3429	1,787 / 2,889
3988	0	0	0	0	0	223	0	0	0	0	0	0	0	0	0.1659	5,338 / 3,888
2396	0	0	0	0	0	0	2348	0	0	0	0	0	0	0	0.5292	2,336 / 4,344
483	0	0	0	312	1	0	0	3688	0	0	0	0	0	0	0.1794	483 / 4,344
8344	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	6,344 / 8,344
8344	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	6,344 / 8,344
9178	0	0	0	0	0	0	0	0	0	0	0	147	0	0	0.1734	5,378 / 3,123
9178	0	0	0	0	0	0	0	0	0	0	0	0	988	0	0.1424	5,378 / 8,344
9832	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	5,832 / 5,832
229889	2291	2711	0	4257	489	281	4296	8987	0	0	0	284	1211	0	0.2481	56,056 / 229,472

Fig. 16 Confusion matrix for test data set

Illustration 2: Naïve Bayes Classification in H2O

The same principle as with the Random Forest model is applied to the Naïve Bayes model. The “label” column selected as the response column as shown in Fig. 17. The output metrics on the training model is shown in Fig. 18.

Table 5 Analysis of percentage difference between predicted and already known “label”

Number of training data	982,273
Number of subjects	10
Number of test data	233,472
Correct predictions	177,416
Incorrect predictions	56,056
Percentage accuracy	75.99%



Fig. 17 Building a Naïve Bayes model

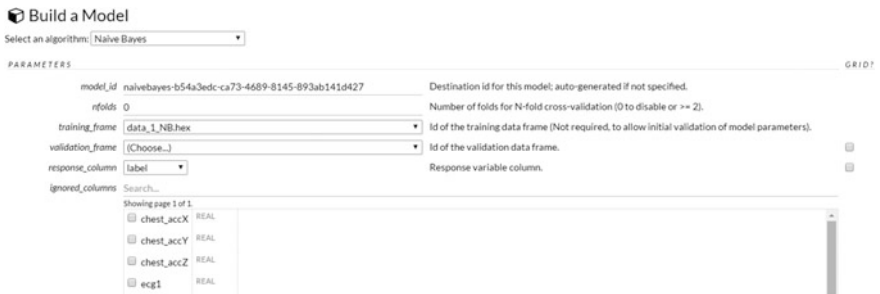


Fig. 18 Output metrics for training model

Figure 19 shows the step where the model is used to predict the “label” for the test set.

The predicted “label” can be merged with the test data set and compared with that already known. Figure 20 shows the confusion matrix for the test data set.

An analysis of the percentage difference between the predicted and already known “label” for the test data set can be performed as shown in Table 6.

Illustration 3: Deep Learning Classification in H2O

The same principle as with the Random Forest model and Naïve Bayes is applied to the Deep Learning model. The “label” column selected as the response column as shown in Fig. 21. The output metrics on the training model is shown in Fig. 22.



Fig. 19 Prediction using training model on test data set

• PREDICTION - CM - CONFUSION MATRIX

Full	cloning_status	cutting	fronze_situation_ones	jugging	jugg_start_end_block	items_bending	tying_knaw	running	stirring_end_relaving	standing_start11	soort_beked_forward	soiling	fronze_situation_ones	Error Rate
58643	4882	3218	0	0	0	0	0	0	0	0	0	0	0	0.4395 185,785 / 245,428
1712	3290	0	0	12	0	0	783	0	2	0	0	0	0	0.4482 1,794 / 4,004
84	1	8071	0	0	0	0	0	0	0	0	0	0	0	0.5819 73 / 125,344
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5 / 0
809	0	0	0	4064	485	0	0	183	0	0	0	0	1	0.5289 1,483 / 2,824
879	0	0	0	1074	416	0	0	29	0	0	0	0	1	0.5818 1,083 / 1,890
3	36	44	43	0	0	0	484	0	0	0	0	0	0	0.5129 1,242 / 2,420
0	0	0	0	25	0	0	4114	1	0	0	0	0	0	0.5350 39 / 72,844
27	0	0	0	1181	429	0	0	4383	0	0	0	0	0	0.5252 1,962 / 3,744
0	0	0	0	199	0	0	0	0	0	0	0	0	0	0.5177 189 / 3,644
0	2	0	1	0	0	0	0	0	0	0	0	0	0	0.5393 188 / 3,544
189	17	0	143	0	0	0	183	0	0	0	0	0	0	0.5187 718 / 1,323
148	842	0	0	0	0	0	79	0	0	0	0	0	0	0.5210 1,375 / 2,644
28	0	0	1196	0	0	0	0	0	0	0	0	0	0	0.510 8,812 / 17,412
8122	8880	8311	0	0	0	0	0	0	0	0	0	0	0	0.5253 112,818 / 213,472

Fig. 20 Confusion matrix for test data set

Table 6 Analysis of percentage difference between predicted and already known “label”

Number of training data	982,273
Number of subjects	10
Number of test data	233,472
Correct predictions	176,202
Incorrect predictions	57,270
Percentage accuracy	75.47%



Fig. 21 Building a deep learning model

```

▼ OUTPUT - TRAINING_METRICS
  model      deeplearning-aa9a026f-b38e-4c39-a088-ce105d058ef3
  model_checksum -8629438079940385792
  frame      .
  frame_checksum 0
  description Metrics reported on temporary training frame with 10032 samples
  model_category Multinomial
  scoring_time 1487415393998
  predictions .
  MSE        0.116052
  RMSE       0.340664
  nobs       10032
  r2         0.990465
  logloss    0.381634
  mean_per_class_error 0.335862

```

Fig. 22 Output metrics for training model

```

predict

Predict
Name: prediction-00b54795-78cf
Model: deeplearning-aa9a026f-b38e-4c39-a088-ce105d058ef3
Frame: data_2_NB.hex
Action: Predict

predict model: "deeplearning-aa9a026f-b38e-4c39-a088-ce105d058ef3", frame: "data_2_NB.hex", predictions_frame: "prediction-00b54795-78cf-4c1a-ad13-b0c8080dffa3"

```

Fig. 23 Prediction using training model on test data set

• PREDICTION - CM - CONFUSION MATRIX

Null	c1500g_stairs	cycling	frontal_elevation_arms	jogging	jump_front_and_back	kneels_bending	lying_down	running	sitting_and_reclining	standing_still	wallet_bands	wardrobe	frontal_elevation_arms	Error	Rate
102300	9234	665	0	1262	375	879	1679	2871	18	402	42	1708	0	0.0790	10,000
1847	4277	7	0	0	0	3	1	0	0	0	0	0	0	0.0000	149,428
2386	0	3758	0	0	0	0	0	0	0	0	0	0	0	0.0000	3,187 / 8,144
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 0
1290	0	0	0	4811	22	0	0	2	0	0	0	0	0	0.2130	3,113 / 8,144
1480	3	0	0	142	470	2	0	2	0	0	0	0	0	0.1760	3,020 / 2,009
9174	134	0	0	0	0	928	0	0	0	0	0	0	0	0.0000	3,208 / 8,100
3072	0	0	0	0	0	0	3072	0	0	0	0	0	0	0.0000	3,072 / 8,144
383	0	0	0	5	125	0	0	5811	0	0	0	0	0	0.0000	913 / 8,144
8144	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	8,144 / 8,144
8144	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	8,144 / 8,144
9325	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	9,325 / 9,325
2485	1580	0	0	0	0	0	0	0	0	0	0	2063	0	0.0640	4,062 / 8,144
5621	0	0	0	0	0	0	0	0	0	11	0	0	0	1.0	9,932 / 9,932
103727	9234	4430	0	8240	887	1412	4782	8308	18	443	42	3871	0	0.2428	58,478 / 239,472

Fig. 24 Confusion matrix for test data set

Figure 23 shows the step where the model is used to predict the “label” for the test set.

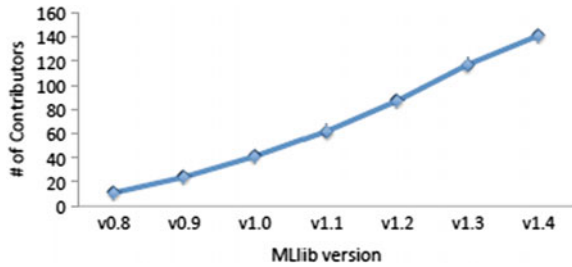
The predicted “label” can be merged with the test data set and compared with that already known. Figure 24 shows the confusion matrix for the test data set.

An analysis of the percentage difference between the predicted and already known “label” for the test data set can be performed as shown in Table 7.

Table 7 Analysis of percentage difference between predicted and already known “label”

Number of training data	982,273
Number of subjects	10
Number of test data	233,472
Correct predictions	176,996
Incorrect predictions	56,476
Percentage accuracy	75.81%

Fig. 25 Growth in MLlib contributors [55]



1.2.2 MLlib

Spark MLlib is an open source API that is part of the Apache Software Foundation. Spark MLlib and DataFrames provide several tools. These tools ease the integration of existing workflows developed on tools such as R and Python, with Spark. For example, users are provided with possibility to use the R syntax they are familiar with and call the appropriate MLlib algorithms [53].

The UC Berkeley AMPLab pioneered Spark and the latter was made open-source in 2010. The major aim for the design of Spark is efficient iterative computing. Machine learning algorithms have also been packaged with the early releases of Spark. With the creation of MLlib, the lack of scalable and robust learning algorithms was overcome. As part of the MLbase project [54], MLlib was developed in 2012 and was made open-source in September 2013. When MLlib was first launched, it was incorporated in Spark’s version 0.8. The Apache 2.0 license packages Spark and MLlib as an open-source Apache project. Moreover, as of Spark’s version 1.0, Spark and MLlib have been released on a three month cycle [5].

Eleven contributors helped developing the initial version of MLlib at UC Berkeley. The original version provided only a limited group of common machine learning algorithms. With the initiation of MLlib, the number of contributors towards the project has grown exponentially. The number of contributors as from the Spark 1.4 release has grown to more than 140 contributors from more than 50 organizations in a time span of less than two years. Figure 25 depicts the number of contributors of MLlib against the release versions. The implementation of several other functionalities was motivated by the adoption rate of MLlib [55].

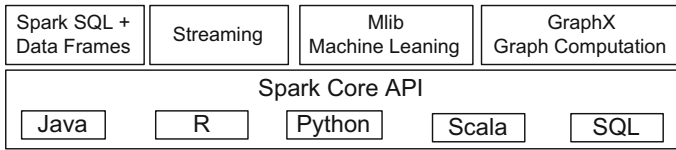


Fig. 26 The spark ecosystem

Architecture and Features

Several components within the Spark ecosystem are beneficial for MLlib. Figure 26 illustrates the Spark ecosystem.

As observed in Fig. 25, Spark supports programming platforms such as R, SQL, Python, Scala and Java. Additionally, it has several libraries which can provide functionalities such as graph computations, stream data processing, and real-time interactive query processing in addition to machine learning [53].

A general execution engine is provided for data transformation is provided by Spark core at the very low level. These transformations involve more than 80 operators for performing feature extraction and data cleaning. There are other high-level libraries included in Spark and leveraged by MLlib. The functionality of data integration is provided by Spark SQL which aid in the simplification of data preprocessing and cleaning. Support for the fundamental DataFrame abstraction in Spark.ml package is also provided.

Large-scale graph processing is supported by GraphX [56]. The implementation of learning algorithms that can be modeled as large, sparse graph problems, e.g., LDA [57, 58] is also supported. Additionally, Spark Streaming [59] provides the ability to handle and process live data streams to users, thereby enabling the development of online learning algorithms, as in Freeman [60]. The improvements in MLlib are brought up by performance enhancements of the high-level libraries in Spark core.

E-client distributed learning and prediction are supported by the several optimizations included in MLlib. A careful use of blocking is made by the ALS algorithm for recommendations in order to lower the collection of JVM garbage overhead and to leverage high-level operations of linear algebra. Ideas such as data-dependent feature discretization for the reduction of communication costs, and parallelized learning within and across trees using ensembles of trees have been used by Decision trees from the PLANET project [61]. Optimization algorithms are used to learn the Generalized Linear Models. These algorithms perform the gradient computation in parallel by making use of c++ based linear algebra libraries for worker computations. E-client communication primitives are very beneficial for many algorithms. The driver is prevented from being a bottleneck by tree-structured aggregation, and large models are quickly distributed to workers by Spark broadcasts.

Performing machine learning in practice involves a set of stages in an appropriate sequence. The stages in the proper order are: pre-processing of data,

extraction of features, fitting of models, and validation. Native support for the multiple set of functionalities needed for the construction of the pipeline are not provided by most of the machine learning libraries. The process of putting together an end-to-end pipeline has severe implications in terms of cost and labor for the network overhead when dealing with large-scale datasets. A package is added in MLlib with the aim to address these concerns inspired by the previous works of [62–64], and [65], and by leveraging on Spark’s rich ecosystem. The implementation and setting of cascaded learning pipelines is supported by the spark.ml package via a standard set of high-level APIs [66].

In order to ease the combination of several algorithms in a single workflow, the APIs for machine learning algorithms have been standardized by MLlib. The Scikit-Learn project has brought forward the pipeline concept. The key concepts introduced by the pipelines API are: DataFrame, Transformer, Estimator, Pipeline and Parameter. These are discussed next.

DataFrame:

DataFrame from the Spark SQL is used by the ML API as an ML dataset. The dataset can contain several data types and a DataFrame can consist of several columns comprising of feature vectors, text, labels, and predicted labels.

Transformer:

A Transformer is an algorithm with the capability of acting upon a DataFrame and converting it to another DataFrame. A machine learning model, for example, is a Transformer which converts a DataFrame consisting of features into predictions.

Estimator:

The role of an Estimator algorithms is to learn from the training DataFrame and produce a mathematical model. A Transformer is produced when an Estimator is fit on a DataFrame.

Pipeline:

The role of a Pipeline is to concatenate several Estimators and Transformers which specify a workflow.

Parameter:

The parameters specified for all Transformers and Estimators share a common API [67]. In machine learning, it is common to run a sequence of algorithms to process and learn from data. For example, a simple text document processing workflow might include the following stages:

- Splitting of each document’s text into words.
- Converting each word into a numerical feature vector.
- Learning of a prediction model by making use of the feature vectors and labels.

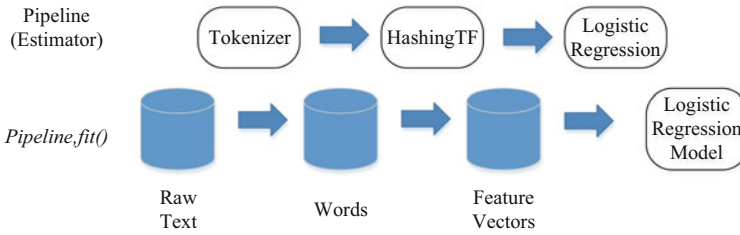


Fig. 27 Training time usage of a pipeline

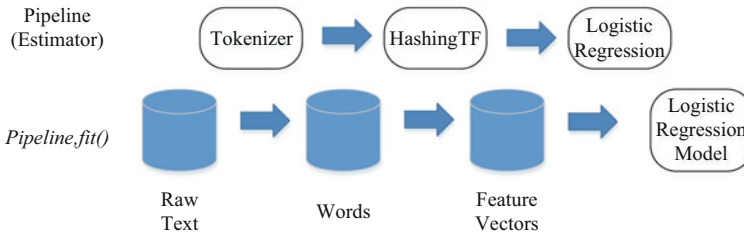


Fig. 28 Pipeline transformer

This type of workflow is represented as a Pipeline by MLlib. The Pipeline consists of a sequence of Estimators and Transformers to be executed in a predefined and specific order. A transformation operation is performed on the input DataFrame as it flows through each stage of the Pipeline. The *transform()* method is called on the DataFrame at the Transformer stages while the *fit()* method is called on the DataFrame for the Estimator stages. Consider for example the simple text document workflow. Figure 27 illustrates the *training time* usage of a Pipeline [67].

A Pipeline with three stages are depicted in Fig. 27. The first and the second stages (Tokenizer and HashingTF) are Transformers and the third stage (Logistic Regression) is an Estimator. The data flow through the Pipeline is represented in Fig. 27 and the cylinders indicate DataFrames. The Pipeline.fit() method is applied on the original DataFrame comprising of the raw text documents and labels. The Tokenizer.transform() method performs a splitting on the raw text documents to convert them into words and adding a new column to the DataFrame with the words. The HashingTF.transform() method performs a conversion operation on the words to form feature vectors, thereby adding a new column to the DataFrame with these vectors. The Pipeline calls LogisticRegression.fit() to produce the machine learning model since LogisticRegression is an Estimator. Further stages of the Pipeline would call the model's transform() method on the DataFrame before passing the latter to the other stage [67]. A Pipeline is an Estimator which produces a Transformer following the running of the Pipeline's fit() method. Figure 28 depicts the Pipeline model used for the test phase.

Figure 28 shows the Pipeline model which consists of the same number of stages as the original Pipeline with the difference that all the Estimators have become

Transformers in this case. The model's *transform()* method is executed on the test dataset and the dataset is updated at each stage before being passed to the next. The Pipeline and models aid in ensuring that the training and test data undergo the same feature processing steps [67].

MLlib Algorithms

The different categories of algorithms supported by MLIB are as follows:

A. Extracting, transforming and selecting features:

The algorithms for working with features are roughly divided into the following groups [68]:

- Extraction: Extracting features from “raw” data.
- Transformation: Scaling, converting, or modifying features.
- Selection: Selecting a subset from a larger set of features.
- Locality Sensitive Hashing (LSH): This class of algorithms combines aspects of feature transformation with other algorithms.

B. Classification and regression:

The classification and regression algorithms are given as follows [69]:

Classification:

- Naive Bayes
- Decision tree classifier
- Random forest classifier
- One-vs-rest classifier
- Multilayer perceptron classifier
- Gradient-boosted tree classifier.

Regression:

- Isotonic regression
- Logistic regression
 - Binomial logistic regression
 - Multinomial logistic regression
- Linear regression
- Generalized linear regression
- Survival regression
- Decision tree regression
- Random forest regression
- Gradient-boosted tree regression.

The Clustering algorithms are given as follows [70]:

- K-means
- Bisecting k-means
- Latent Dirichlet allocation (LDA)
- Gaussian Mixture Model (GMM).

MLIB also supports function for Collaborative Filtering [71], ML Tuning: model selection and hyperparameter tuning [72].

MLlib Deployment and Application Example

MLlib can be used locally in a Java programming IDE like Eclipse by importing the corresponding jar file in the workspace. In this work, the environment already available in the online Databricks community edition has been used [53]. Users simply need to sign up and use the platform.

In this section, the Mobile Health (MHEALTH) dataset from [50–52] has been used to demonstrate classification machine learning algorithms with MLIB.

The classification algorithms used in this section are: Random Forest, Decision Tree, and Multinomial Logistic Regression [36]. Naïve Bayes is not used in this case because the current version does not support negative values for input features. The sensors in the case of the MHEALTH dataset [51, 52, 73] output both positive and negative values. The common dataset to be used with the classification algorithms can be imported using the *Create Table* option on the Databricks homepage as depicted in Fig. 29.

Given that the community edition is being use, there is a limitation on the resources provided which would not allow the modelling using the large data set as with H2O on the local machine. Therefore, part of the MHEALTH dataset is uploaded to Databricks for the presentation of the modelling with the different classification algorithms.

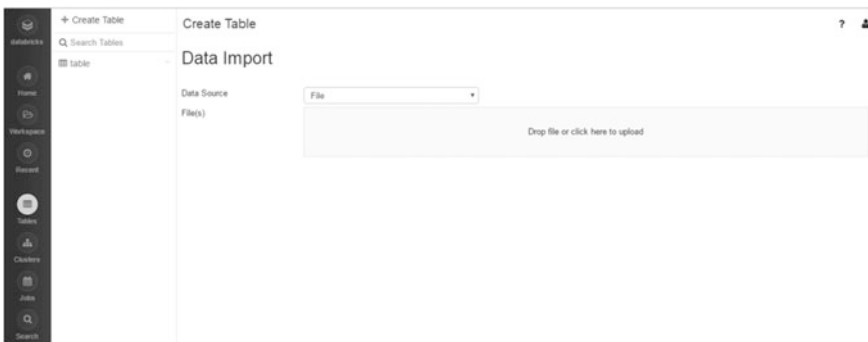


Fig. 29 Import data set on the cluster to be processed in databricks

```
import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
import org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel

val data = sc.textFile("/FileStore/tables/71xvheio1487641043634/data_2_mhealth.csv", 6)

import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
import org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
data: org.apache.spark.rdd.RDD[String] = /FileStore/tables/71xvheio1487641043634/data_2_mhealth.csv RapPartitionsRDD[1142] at textFile at <console>:71
Command took 9.12 seconds -- by yjgash316@hotmail.com at 2/23/2017, 7:05:13 PM on My Cluster
```

Fig. 30 Import the libraries

```
> //To find the headers
val header = data.first;

//To remove the header
val data1 = data.filter(_(0) != header(0));

> {} Spark Jobs
header: String = chest_acc1,chest_accy,chest_accz,ecg1,ecg2,la_accX,la_accY,la_accZ,la_gyroX,la_gyroY,la_gyroZ,la_mag1,la_magY,la_magZ,rla_accX,rla_accY,rla_accZ,rla_gyroX,rla_gyroY,rla_gyroZ,rla_mag1,rla_magY,rla_magZ,movement
data1: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[499] at filter at <console>:62
Command took 8.61 seconds -- by yjgash316@hotmail.com at 2/23/2017, 6:58:22 PM on My Cluster
```

Fig. 31 Filter header from RDD

The MLlib algorithms used in this section require the input data to be in the form of a Resilient Distributed Dataset (RDD) of LabeledPoints. The way the data is read needs to be very specific to be able to proceed with the modelling. Figure 30 shows all the necessary library imports to be used and the Scala code to read data from the csv file as an RDD. The headers are located and removed so as to obtain only the numerical data in the RDD and further processing and application of Machine Learning algorithms as shown in Fig. 31. The data has to be parsed to obtain the label and feature vectors (RDD of LabeledPoints) as shown in Fig. 32. The data is then split into the training and test sets to be used for each classification model. The training and test sets are given 60 and 40% of the dataset respectively. The allocation is done using a random shuffling as shown in Fig. 33.

This process is identical for all the algorithms shown in the following sub-sections.

Illustration 1: Random Forest Classification with MLlib and Spark on Databricks

The model is then built using the training data set as input as shown in Fig. 34.

With the training model obtained, prediction can be performed on the test set data. A comparison can then be performed on the predicted and already known

```
> val parsedData = data.map( line => {  
  val parts = line.split(',')  
  LabeledPoint(parts(parts.length-1).toDouble, Vectors.dense(parts.slice(0, parts.length-2).map(_._toDouble)))  
})  
  
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[501] at map at <console>:62  
Command took 0.38 seconds -- by yogesh336@hotmail.com at 2/23/2017, 6:58:28 PM on My Cluster
```

Fig. 32 Parse RDD to obtain RDD of LabeledPoints with label and feature vector

```
> val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)  
val training = splits(0)  
val test = splits(1)  
  
splits: Array[org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]] = Array(MapPartitionsRDD[502] at randomSplit at <console>:70, MapPartitionsRDD[503] at randomSplit at <console>:70)  
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[502] at randomSplit at <console>:70  
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[503] at randomSplit at <console>:70  
Command took 0.22 seconds -- by yogesh336@hotmail.com at 2/23/2017, 6:58:31 PM on My Cluster
```

Fig. 33 Split into training and test sets

```
> val numClasses = 13  
val categoricalFeaturesInfo = Map[Int, Int]()  
val numTrees = 10 // Use more in practice.  
val featureSubsetStrategy = "auto" // Let the algorithm choose.  
val impurity = "gini"  
val maxDepth = 20  
val maxBins = 32  
  
val modelRF = RandomForest.trainClassifier(parsedData, numClasses, categoricalFeaturesInfo, numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)  
  
▶ (23) Spark Jobs  
numClasses: Int = 13  
categoricalFeaturesInfo: scala.collection.immutable.Map[Int,Int] = Map()  
numTrees: Int = 10  
featureSubsetStrategy: String = auto  
impurity: String = gini  
maxDepth: Int = 20  
maxBins: Int = 32  
modelRF: org.apache.spark.mllib.tree.model.RandomForestModel =  
TreeEnsembleModel classifier with 10 trees  
  
Command took 1.24 minutes -- by yogesh336@hotmail.com at 2/23/2017, 6:58:35 PM on My Cluster
```

Fig. 34 Training of random forest model

“label”. Figure 35 shows the step where the model is used to predict the “label” for the test set and compute the test error. The computation of the mean squared error is shown in Fig. 36.

The confusion matrix can also be obtained as shown in Fig. 37.

The overall statistics can be computed as shown in Fig. 38.

The precision by label can be obtained as shown in Fig. 39.

The recall by label can be obtained as shown in Fig. 40.

The false positive rate by label can be obtained as shown in Fig. 41.


```

> val labelAndPreds = test.map { point =>
  val prediction = modelRF.predict(point.features)
  (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / test.count()
println("Test Error = " + testErr)
println("Learned classification forest model:\n" + modelRF.toDebugString)

```

▶ (2) Spark Jobs

```

Test Error = 0.028756228543139722
Learned classification forest model:
TreeEnsembleModel classifier with 10 trees

```

Fig. 35 Prediction using training model on the test data set

```

> val labelAndPreds = test.map { point =>
  val prediction = modelRF.predict(point.features)
  (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / test.count()
println("Test Error = " + testErr)
println("Learned classification forest model:\n" + modelRF.toDebugString)

```

▶ (2) Spark Jobs

```

Test Error = 0.028756228543139722
Learned classification forest model:
TreeEnsembleModel classifier with 10 trees

```

Fig. 36 Computation of the mean squared error

```

// Instantiate metrics object
val metrics = new MulticlassMetrics(labelAndPreds)

// Confusion matrix
println("Confusion matrix:")
println(metrics.confusionMatrix)

```

▶ (2) Spark Jobs

```

Confusion matrix:
63353.0 17.0 26.0 10.0 7.0 3.0 12.0 12.0 ... (13 total)
455.0 2427.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
283.0 0.0 2428.0 0.0 0.0 0.0 0.0 0.0 ...
362.0 0.0 0.0 2441.0 0.0 0.0 0.0 0.0 ...
330.0 0.0 0.0 0.0 2447.0 0.0 0.0 0.0 ...
4.0 0.0 0.0 0.0 0.0 2437.0 0.0 0.0 ...
141.0 0.0 0.0 0.0 0.0 0.0 2066.0 0.0 ...
173.0 0.0 0.0 0.0 0.0 0.0 0.0 2219.0 ...
287.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
198.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
197.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
128.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
15.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
metrics: org.apache.spark.mllib.evaluation.MulticlassMetrics = org.apache.spark.mllib.evaluation.MulticlassMetrics@38d05f87
Command took 11.14 seconds -- by yogesh536@hotmail.com at 2/23/2017, 7:45:52 PM on My Cluster

```

Fig. 37 Confusion matrix

```
> // Overall Statistics
val accuracy = metrics.accuracy
println("Summary Statistics")
println(s"Accuracy = $accuracy")

▶ (1) Spark Jobs
Summary Statistics
Accuracy = 0.9712437714568602
accuracy: Double = 0.9712437714568602
Command took 5.83 seconds -- by yogesh536@hotmail.com at 2/23/2017, 7:46:23 PM on My Cluster
```

Fig. 38 Overall statistics

```
> // Precision by label
val labels = metrics.labels
labels.foreach { l =>
  println(s"Precision($l) = " + metrics.precision(l))
}

▶ (1) Spark Jobs
Precision(0.0) = 0.9609713921669751
Precision(1.0) = 0.9930441898527005
Precision(2.0) = 0.9894059529747351
Precision(3.0) = 0.9959200326397388
Precision(4.0) = 0.9971475142624286
Precision(5.0) = 0.9987704918032787
Precision(6.0) = 0.9942252165543792
Precision(7.0) = 0.994621246077992
Precision(8.0) = 0.9969578444154715
Precision(9.0) = 1.0
Precision(10.0) = 0.9995840266222962
Precision(11.0) = 0.9992076069730587
Precision(12.0) = 0.9975278121137207
labels: Array[Double] = Array(0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0)
Command took 10.43 seconds -- by yogesh536@hotmail.com at 2/23/2017, 7:46:47 PM on My Cluster
```

Fig. 39 Precision by label

Illustration 2: Decision Tree Classification with MLlib and Spark on Databricks

The model is built using the training data set as input as shown in Fig. 42.

With the training model obtained, prediction can be performed on the test set data. A comparison can then be performed on the predicted and already known “label”. Figure 43 shows the step where the model is used to predict the “label” for the test set and compute the test error. The computation of the mean squared error is shown in Fig. 44.

The confusion matrix can also be obtained as shown in Fig. 45.

The overall statistics can be computed as shown in Fig. 46.

The precision by label can be obtained as shown in Fig. 47.

```
> // Recall by label
labels.foreach { l =>
  println(s"Recall($l) = " + metrics.recall(l))
}
```

Recall(0.0) = 0.9984397654920255
Recall(1.0) = 0.8421235253296322
Recall(2.0) = 0.8956104758391737
Recall(3.0) = 0.8708526578665715
Recall(4.0) = 0.8811667266834714
Recall(5.0) = 0.9983613273248668
Recall(6.0) = 0.9361123697326688
Recall(7.0) = 0.927675585284281
Recall(8.0) = 0.8888027896164278
Recall(9.0) = 0.9238754325259516
Recall(10.0) = 0.9242307692307692
Recall(11.0) = 0.9516981132075472
Recall(12.0) = 0.9817518248175182

Fig. 40 Recall by label

```
> // False positive rate by label
labels.foreach { l =>
  println(s"FPR($l) = " + metrics.falsePositiveRate(l))
}
```

FPR(0.0) = 0.08731801676451624
FPR(1.0) = 1.8881126647933628E-4
FPR(2.0) = 2.8822277403334516E-4
FPR(3.0) = 1.109680855785876E-4
FPR(4.0) = 7.765525504204478E-5
FPR(5.0) = 3.315723159221026E-5
FPR(6.0) = 1.322867977775818E-4
FPR(7.0) = 1.3255713764954101E-4
FPR(8.0) = 7.74867719010826E-5
FPR(9.0) = 0.0
FPR(10.0) = 1.1071867491889857E-5
FPR(11.0) = 2.2156000398808007E-5
FPR(12.0) = 2.17162339707048E-5

Fig. 41 False positive rate by label

```

> // Train a DecisionTree model.
// Empty categoricalFeaturesInfo indicates all features are continuous.
val numClasses = 13
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32

val modelDT = DecisionTree.trainClassifier(parsedData, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins)

▶ (8) Spark Jobs
numClasses: Int = 13
categoricalFeaturesInfo: scala.collection.immutable.Map[Int,Int] = Map()
impurity: String = gini
maxDepth: Int = 5
maxBins: Int = 32
modelDT: org.apache.spark.mllib.tree.model.DecisionTreeModel = DecisionTreeModel classifier of depth 5 with 63 nodes

```

Fig. 42 Training of decision tree model

```

> // Evaluate model on test instances and compute test error
val labelAndPreds = test.map { point =>
  val prediction = modelDT.predict(point.features)
  (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / test.count()
println("Test Error = " + testErr)
println("Learned classification tree model:\n" + modelDT.toDebugString)

▶ (2) Spark Jobs
Test Error = 0.20314467439382689
Learned classification tree model:
DecisionTreeModel classifier of depth 5 with 63 nodes

```

Fig. 43 Prediction using training model on the test data set

```

> val testMSE = labelAndPreds.map{ case(v, p) => math.pow((v - p), 2)}.mean()
println("Test Mean Squared Error = " + testMSE)

▶ (1) Spark Jobs
Test Mean Squared Error = 9.233482925989286
testMSE: Double = 9.233482925989286

```

Fig. 44 Computation of the mean squared error

```

> // Instantiate metrics object
val metrics = new MulticlassMetrics(labelAndPreds)

// Confusion matrix
println("Confusion matrix:")|
println(metrics.confusionMatrix)

```

▸ (2) Spark Jobs

Confusion matrix:

63484.0	19.0	26.0	8.0	1.0	5.0	8.0	7.0	...	(13 total)
361.0	2425.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
257.0	0.0	2428.0	0.0	0.0	0.0	0.0	0.0	...	
334.0	0.0	0.0	2443.0	0.0	0.0	0.0	0.0	...	
333.0	0.0	0.0	0.0	2453.0	0.0	0.0	0.0	...	
2.0	0.0	0.0	0.0	0.0	2435.0	0.0	0.0	...	
115.0	0.0	0.0	0.0	0.0	0.0	2070.0	0.0	...	
168.0	0.0	0.0	0.0	0.0	0.0	0.0	2224.0	...	
239.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
230.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
237.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
159.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

Fig. 45 Confusion matrix

Fig. 46 Overall statistics

```

> // Overall Statistics
val accuracy = metrics.accuracy
println("Summary Statistics")
println(s"Accuracy = $accuracy")

```

▸ (1) Spark Jobs

Summary Statistics
Accuracy = 0.972761222139713
accuracy: Double = 0.972761222139713

The recall by label can be obtained as shown in Fig. 48.

The false positive rate by label can be obtained as shown in Fig. 49.

Illustration 3: Multinomial Logistic Regression Classification with MLlib and Spark on Databricks

The model is built using the training data set as input as shown in Fig. 50. With the training model obtained, prediction can be performed on the test set data. A comparison can then be performed on the predicted and already known “label”.

```

> // Precision by label
val labels = metrics.labels
labels.foreach { l =>
  println(s"Precision($l) = " + metrics.precision(l))
}

▶ (1) Spark Jobs
Precision(0.0) = 0.9629584685859903
Precision(1.0) = 0.9922258592471358
Precision(2.0) = 0.9894050529747351
Precision(3.0) = 0.9967360261117911
Precision(4.0) = 0.9995925020374898
Precision(5.0) = 0.9979508196721312
Precision(6.0) = 0.9961501443695862
Precision(7.0) = 0.9968623935454953
Precision(8.0) = 0.9973924380704041
Precision(9.0) = 1.0
Precision(10.0) = 1.0
Precision(11.0) = 0.998811410459588
Precision(12.0) = 0.992583436341162
Labels: Array[Double] = Array(0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0)

```

Fig. 47 Precision by label

```

> // Recall by label
labels.foreach { l =>
  println(s"Recall($l) = " + metrics.recall(l))
}

Recall(0.0) = 0.8309440462299272
Recall(2.0) = 0.6726342710997443
Recall(3.0) = 0.7667610953729934
Recall(5.0) = 0.49122807017543857
Recall(6.0) = 0.7913513513513514
Recall(7.0) = 0.6051660516605166
Recall(8.0) = 0.640325865580448
Recall(9.0) = 0.8121845710165826
Recall(10.0) = 0.6209311907704985
Recall(11.0) = 0.5853827502597853

```

Fig. 48 Recall by label

The confusion matrix can also be obtained as shown in Fig. 51.
 The overall statistics can be computed as shown in Fig. 52.
 The precision by label can be obtained as shown in Fig. 53.
 The recall by label can be obtained as shown in Fig. 54.
 The false positive rate by label can be obtained as shown in Fig. 55.
 The F-measure by label can be computed as shown in Fig. 56.
 The weighted statistics can be computed as shown in Fig. 57.

```
> // False positive rate by label
labels.foreach { l =>
  println(s"FPR($l) = " + metrics.falsePositiveRate(l))
}

FPR(0.0) = 0.2918159667478859
FPR(2.0) = 9.731543624161074E-4
FPR(3.0) = 1.6714581801163334E-4
FPR(5.0) = 0.025974025974025976
FPR(6.0) = 0.014631389003630672
FPR(7.0) = 0.006551452737531732
FPR(8.0) = 0.008058454191722674
FPR(9.0) = 0.001663985800654501
FPR(10.0) = 0.009912478451133801
FPR(11.0) = 0.009263373022925183
```

Fig. 49 False positive rate by label

```
// Run training algorithm to build the model
val modelLRBFOS = new LogisticRegressionWithLBFGS()
  .setNumClasses(13)
  .run(training)

// Compute raw scores on the test set
val predictionAndLabels = test.map { case LabeledPoint(label, features) =>
  val prediction = modelLRBFOS.predict(features)
  (prediction, label)
}

(S0) Spark Jobs
modelLRBFOS: org.apache.spark.mllib.classification.LogisticRegressionModel = org.apache.spark.mllib.classification.LogisticRegressionModel: Intercept = 0.0, nu
nFeatures = 264, numClasses = 13, threshold = 0.5
predictionAndLabels: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[441] at map at <console>:65
```

Fig. 50 Training of decision tree model

1.2.3 Comparative Analysis of Machine Learning Algorithms

The Machine Learning techniques used in this work together with the open source tools are: Naïve Bayes, Random Forest, Deep Learning, Decision Tree, and Multinomial Logistic Regression. The description, advantages, and limitations are given in Table 8.

1.3 Conclusions and Discussions

The aim of this Chapter was to present the various analytic tools which can be used for the processing of big data, and machine learning tools for analysing and

```

> // Instantiate metrics object
val metrics = new MulticlassMetrics(predictionAndLabels)

// Confusion matrix
println("Confusion matrix:")
println(metrics.confusionMatrix)

▶ (2) Spark Jobs
Confusion matrix:
58887.0  879.0  423.0  890.0  321.0  140.0  475.0  543.0  ... (13 total)
1488.0  956.0  0.0  0.0  0.0  0.0  0.0  0.0  ...
2454.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...
0.0  0.0  0.0  2451.0  0.0  0.0  0.0  0.0  ...
2356.0  4.0  0.0  0.0  94.0  0.0  0.0  0.0  ...
2209.0  13.0  0.0  0.0  35.0  30.0  5.0  15.0  ...
790.0  0.0  0.0  0.0  0.0  0.0  1287.0  0.0  ...
1470.0  0.0  1.0  0.0  0.0  0.0  0.0  760.0  ...
998.0  0.0  0.0  0.0  0.0  0.0  10.0  0.0  ...
59.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  ...
1177.0  0.0  10.0  0.0  0.0  0.0  0.0  15.0  ...
1095.0  6.0  79.0  0.0  52.0  26.0  4.0  2.0  ...
339.0  37.0  5.0  3.0  84.0  0.0  29.0  30.0  ...

```

Fig. 51 Confusion matrix

Fig. 52 Overall statistics

```

> // Overall Statistics
val accuracy = metrics.accuracy
println("Summary Statistics")
println(s"Accuracy = $accuracy")

▶ (1) Spark Jobs
Summary Statistics
Accuracy = 0.7597584993381332
accuracy: Double = 0.7597584993381332

```

obtaining insights from big data. Essentially, the use of open-source tools: H2O and Spark MLlib, by developing classification based machine learning algorithms for M-Health data, has been illustrated. Random Forest, Naïve Bayes, and Deep Learning algorithms have been used on the data in H2O. Random Forest, Decision Trees, and Multinomial Logistic Regression Classification algorithms have been used with the data in Spark MLlib. The main contribution in this Chapter is that the illustrations demonstrate the flows for developing, training, and testing


```

> // Precision by label
val labels = metrics.labels
labels.foreach { l =>
  println(s"Precision($l) = " + metrics.precision(l))
}

▶ (1) Spark Jobs
Precision(0.0) = 0.8031286653391888
Precision(1.0) = 0.5044854881266491
Precision(2.0) = 0.0
Precision(3.0) = 0.7329545454545454
Precision(4.0) = 0.16040955631399317
Precision(5.0) = 0.15306122448979592
Precision(6.0) = 0.7110497237569061
Precision(7.0) = 0.5563689604685212
Precision(8.0) = 0.6561224489795918
Precision(9.0) = 0.7551679586563308
Precision(10.0) = 0.4979114452798663
Precision(11.0) = 0.5974963890226288
Precision(12.0) = 0.20903954802259886
Labels: Array[Double] = Array(0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0)

```

Fig. 53 Precision by label

```

> // Recall by label
labels.foreach { l =>
  println(s"Recall($l) = " + metrics.recall(l))
}

Recall(0.0) = 0.8932287716530656
Recall(1.0) = 0.3911620294599018
Recall(2.0) = 0.0
Recall(3.0) = 1.0
Recall(4.0) = 0.03830480847595762
Recall(5.0) = 0.012295081967213115
Recall(6.0) = 0.6193455245428297
Recall(7.0) = 0.340654415060511
Recall(8.0) = 0.5588874402433724
Recall(9.0) = 0.9729504785684561
Recall(10.0) = 0.49584026622296173
Recall(11.0) = 0.4916798732171157
Recall(12.0) = 0.09147095179233622

```

Fig. 54 Recall by label

```

> // False positive rate by label
labels.foreach { l =>
  println(s"FPR($l) = " + metrics.falsePositiveRate(l))
}

FPR(0.0) = 0.5347682732560294
FPR(1.0) = 0.010378557612600165
FPR(2.0) = 0.00573702536892721
FPR(3.0) = 0.009870893575628951
FPR(4.0) = 0.005438567401757586
FPR(5.0) = 0.0018346798704671802
FPR(6.0) = 0.005757312226857917
FPR(7.0) = 0.006682251235003529
FPR(8.0) = 0.007437815886468472
FPR(9.0) = 0.008374210084404967
FPR(10.0) = 0.013279566922609513
FPR(11.0) = 0.009248299131589137
FPR(12.0) = 0.0030398436651829334

```

Fig. 55 False positive rate by label

```

> // F-measure by label
labels.foreach { l =>
  println(s"F1-Score($l) = " + metrics.fMeasure(l))
}

F1-Score(0.0) = 0.8457859358841779
F1-Score(1.0) = 0.4406545286932473
F1-Score(2.0) = 0.0
F1-Score(3.0) = 0.8459016393442622
F1-Score(4.0) = 0.0618421052631579
F1-Score(5.0) = 0.022761760242792105
F1-Score(6.0) = 0.6620370370370371
F1-Score(7.0) = 0.422574367528496
F1-Score(8.0) = 0.6036141750762731
F1-Score(9.0) = 0.85033642480451
F1-Score(10.0) = 0.4968736973739058
F1-Score(11.0) = 0.5394479460986742
F1-Score(12.0) = 0.12725709372312985

```

Fig. 56 F-measure by label

```

> // Weighted stats
println(s"Weighted precision: ${metrics.weightedPrecision}")
println(s"Weighted recall: ${metrics.weightedRecall}")
println(s"Weighted F1 score: ${metrics.weightedFMeasure}")
println(s"Weighted false positive rate: ${metrics.weightedFalsePositiveRate}")

Weighted precision: 0.7066478199852897
Weighted recall: 0.7597584993381331
Weighted F1 score: 0.726725431961217
Weighted false positive rate: 0.38160579557566016
    
```

Fig. 57 Weighted statistics

Table 8 Description, advantages, and limitations of machine learning algorithms used

Algorithm	Description	Advantages	Disadvantages/limitations
Naïve Bayes	It is a multiclass classification algorithm which is simple and assumes that every pair of features is independent. It calculates the conditional probability distribution for every feature given label, inside each pass to the training data. After that it employs the Bayes' theorem to obtain the conditional probability distribution of the label given an observation and uses it for prediction	<ul style="list-style-type: none"> - The training of this algorithm can be performed in a very efficient manner - Classifiers based on Naive Bayes have been performed satisfactorily in several non-trivial real-world applications although they rely heavily on assumptions 	<ul style="list-style-type: none"> - The assumption that any feature pair is independent for a specific output class, is very strong for a given data distribution - An estimation for the probability of any value of a feature has to be made using a statistical approach. Consequently, when data available are scarce, probabilistic estimates tend towards either the lower limit i.e. 0 or the upper limit i.e. 1. Fluctuations in numerical results then occur and poor results are obtained
Random forest	Random forests form part of a group of learning techniques for classification, regression and similar related operations. The rationale behind these methods is the construction of multiple decision trees at the training instant. For classification problems they output the mode of	<ul style="list-style-type: none"> - Provides very good accuracy in the classification of several data sets - It executes with good efficiency efficiently on big databases - It can handle Thousands of input variables can be supported deleting any variable 	When applied to specific databases with noisy classification/regression operations, it can lead to an over fitting problem

(continued)

Table 8 (continued)

Algorithm	Description	Advantages	Disadvantages/limitations
	<p>the classes that were being classified. For regression problems, they output the mean prediction or regression of the individual trees</p>		
<p>Deep learning</p>	<p>In Deep learning a concatenation of multiple layers that are capable of non-linear processing are employed for feature extraction and transformation. The output of layer $n - 1$ is fed to layer n as input. As such a hierarchical representation is obtained whereby the features of higher levels are derived from lower levels. Both supervised and unsupervised algorithms can be used for applications such as pattern analysis and classification</p>	<ul style="list-style-type: none"> - Provides significantly better performance for problems in speech recognition, language, vision, and gaming - Decreases the dependence on time consuming processes such as feature engineering - It is equipped with techniques such as convolutional neural networks, recurrent neural networks and long short-term memory. These allow it to easily adapt to problems such as Vision and language 	<ul style="list-style-type: none"> - Huge amounts of data are required so that it really outperform other techniques - The training is very time-consuming and requires major computational costs
<p>Decision tree</p>	<p>The algorithm uses a tree-like model for the representation of classification examples. The tree is learned by performing recursive splits at each node based on a threshold value. The recursive process is halted when a maximum depth is reached or when the condition that splitting a node no longer adds value to predictions is reached</p>	<ul style="list-style-type: none"> - Very little data preparation is required - Statistical tests can be used for the validation of the models - Has very good performance with large volumes of data - Has the capability of handling categorical as well as numerical data types - Interpretation can be done in an easy way 	<ul style="list-style-type: none"> - The trees deviate severely with little changes in training data, which can cause predictions to experience big changes - The accuracy of trees tend to be lower than other approaches - There is always the risk of overfitting data

(continued)

Table 8 (continued)

Algorithm	Description	Advantages	Disadvantages/limitations
Multinomial logistic regression	It is a classification technique which extends the logistic regression mechanism to cases where more than 2 classes need to be dealt with. The model is specifically used to compute and predict the probabilities of the different outcomes/categories based on the dependent features in the training set	<ul style="list-style-type: none"> – There is the possibility of using Ordinary Least Squares (OLS) regression with multiple categorical labels – Categories can be deleted from the set 	<ul style="list-style-type: none"> – The OLS model can make predictions without sense due to the non-continuous nature of the dependent variable – Information, power, and data is lost with deletions as well as causing the samples to be biased

mathematical models in order to obtain insights from M-Health data using open source tools developed for handling big data.

H2O being configured locally, provides the possibility to use a larger dataset as compared to MLlib on Databricks since an online community version of the latter has been used. The M-Health dataset used with H2O has 982,273 data rows which have been used for model training purposes and 233,472 data rows used for model testing purposes. Based on these data-sets, the accuracies of the Random Forest, Naïve Bayes, and Deep Learning models are 75.99, 75.47, and 75.81% respectively.

The MHealth data-set used with MLlib on Databricks consists of 233,472 data rows out of which 70% is used for model training purposes and the remaining used for model testing purposes. Based on these data-sets, the accuracies of the Random Forest, Decision Tree, and Multinomial Logistic Regression models are 97.12, 97.28, and 75.98% respectively.

References

1. Russom, P. (2011). Executive summary: Big data analytics. Renton: The Data Warehouse Institute (TWDI).
2. Landset, S., Khoshgoftaar, T. M., Ritcher, A. M., & Hasanin, T. (2015). A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(24), 1–36.
3. The R Foundation. (2017). *The R project for statistical computing*. Retrieved January 22, 2017 from <https://www.r-project.org/>
4. Machine Learning Group at the University of Waikato Weka 3. (n.d.). *Data mining software in Java*. Retrieved January 22, 2017 from <http://www.cs.waikato.ac.nz/ml/weka/>

5. International Data Corporation. (2017). *Discover the digital universe of opportunities, rich data and the increasing value of the Internet of things*. Retrieved January 22, 2017 from <https://www.emc.com/leadership/digital-universe/index.htm>
6. Vidhya, S., Sarumathi, S., & Shanthi, N. (2014). Comparative analysis of diverse collection of big data analytics tools. *World Academy of Science, Engineering and Technology: International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 8(9), 1646–1652.
7. Kamal, S., Ripon, S. H., Dey, N., Ashour, A. S., & Santhi, V. (2016). A MapReduce approach to diminish imbalance parameters for big deoxyribonucleic acid dataset. *Computer Methods and Programs in Biomedicine*, 131, 161–206.
8. Kamal, S., Dey, N., Ashour, A. S., Ripon, S. H., & Balas, V. E. (2016). FbMapping: An automated system for monitoring Facebook data. *Neural Network World*. doi:10.14311/NNW.2017.27.002
9. Kamal, S., Nimmy, S. F., Hossain, M. I., Dey, N., Ashour, A. S., & Santhi, V. (2016). ExSep: An exon separation process using neural skyline filter. In *International conference on electrical, electronics, and optimization techniques (ICEEOT)*.
10. Bhattacharjee, A., Roy, S., Paul, S., Roy, P., Kaussar, N., & Dey, N. (2016). Classification approach for breast cancer detection using back propagation neural network: A study. In *Biomedical image analysis and mining techniques for improved health outcomes* (p. 12). IGI-Global.
11. Chatterjee, S., Ghosh, S., Dawn, S., Hore, S., & Dey, N. (in press). Optimized forest type classification: A machine learning approach. In *3rd international conference on information system design and intelligent applications*. Vishakhapatnam: Springer AISC.
12. Kimbahune, V. V., Deshpandey, A. V., & Mahalle, P. N. (2017). Lightweight key management for adaptive addressing in next generation internet. *International Journal of Ambient Computing and Intelligence (IJACI)*, 8(1), 20.
13. Najjar, M., Courtemanche, F., Haman, H., Dion, A., & Bauchet, J. (2009). Intelligent recognition of activities of daily living for assisting memory and/or cognitively impaired elders in smart homes. *International Journal of Ambient Computing and Intelligence (IJACI)*, 1(4), 17.
14. IBM. (n.d.). *Big data at the speed of business* [Online]. Retrieved January 22, 2017 from <https://www-01.ibm.com/software/data/bigdata/>
15. Zikopoulos, P., Deroos, D., Parasuraman, K., Deutsch, T., Corrigan, D., & Giles, J. (2013). *Harness the power of big data: The IBM big data platform*. New York: Mc-Graw Hill Companies.
16. IBM. (n.d.). *IBM big data platform* [Online]. Retrieved January 22, 2017 from <https://www-01.ibm.com/software/in/data/bigdata/enterprise.html>
17. IBM. (n.d.). *Bringing big data to the enterprise* [Online]. Retrieved March 12, 2017 from <https://www-01.ibm.com/software/sg/data/bigdata/enterprise.html>
18. Hewlett-Packard Enterprise. (2017). *Big data services: Build an insight engine* [Online]. Retrieved January 22, 2017 from <https://www.hpe.com/us/en/services/consulting/big-data.html>
19. Hewlett-Packard Enterprise. (2017). *Big data software* [Online]. Retrieved March 13, 2017 from <https://saas.hpe.com/en-us/software/big-data-analytics-software>.
20. Hewlett-Packard. (2017). *HAVEN big data platform for developers* [Online]. Retrieved March 15, 2017 from http://www8.hp.com/us/en/developer/HAVEN.html?jumpid=reg_r1002_usen_c-001_title_r0002
21. SAP. (2017). *Unlock business potential from big data more quickly and easily* [Online]. Retrieved January 22, 2017 from <http://www.sap.com/romania/documents/2015/08/f63628c9-3c7c-0010-82c7-eda71af511fa.html>
22. SAP. (2017). *Unlock business potential from your Big Data faster and easier with SAP HANA Vora* [Online]. Retrieved January 22, 2017 from <http://www.sap.com/hk/product/data-mgmt/hana-vora-hadoop.html>

23. SAP Community. (n.d.). *What is SAP HANA?* [Online]. Retrieved March 15, 2017 from <https://archive.sap.com/documents/docs/DOC-60338>
24. Predictive Analytics Today. (n.d.). *50 big data platforms and big data analytics software* [Online]. Retrieved January 22, 2017 from <http://www.predictiveanalyticstoday.com/bigdata-platforms-bigdata-analytics-software/>
25. Microsoft. (2017). *Big data and analytics* [Online]. Retrieved January 22, 2017 from <https://azure.microsoft.com/en-us/solutions/big-data/>
26. Microsoft Developer. (2017). *Microsoft azure services platform* [Online]. Retrieved March 15, 2017 from <https://blogs.msdn.microsoft.com/mikewalker/2008/10/27/microsoft-azure-services-platform/>
27. Microsoft Azure. (2017). *Take a look at these innovative stories by world class companies* [Online]. Retrieved March 24, 2017 from <https://azure.microsoft.com/en-gb/case-studies/>
28. Oracle. (2017). *Big data features* [Online]. Retrieved January 22, 2017 from https://cloud.oracle.com/en_US/big-data/features
29. Oracle. (2017). *Big data in the cloud* [Online]. Retrieved January 22, 2017 from <https://cloud.oracle.com/bigdata>
30. Pollock, J. (2015). *Take it to the limit: An information architecture for beyond Hadoop* [Online]. Retrieved March 24, 2017 from <https://conferences.oreilly.com/strata/big-data-conference-ca-2015/public/schedule/detail/40599>
31. Evosys. (2017). *Oracle big data customers success stories* [Online]. Retrieved March 24, 2017 from <http://www.evosysglobal.com/big-data-2-level>
32. H2O.ai. (n.d.). *Fast scalable machine learning API* [Online]. Retrieved February 17, 2017 from <http://h2o-release.s3.amazonaws.com/h2o/rel-tverberg/4/index.html>
33. The Apache Software Foundation. (2017). *Apache spark MLlib* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/mllib/>
34. LeDell, E. (2015). *High performance machine learning in R with H2O*. Tokyo: ISM HPC on R Workshop.
35. Candel, A., Parmar, V., LeDell, E., & Arora, A. (2016) [Online]. Retrieved February 24, 2017 from <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf>
36. Nykodym, T., & Maj, P. (2017). *Fast analytics on big data with H2O* [Online]. Retrieved February 20, 2017 from http://gotoberlin2014/slides/PetrMaj_and_TomasNykodym_FastAnalyticsOnBigData.pdf
37. Novet, J. (2014). *Oxdata takes \$8.9M and becomes H2O to match its open-source machine-learning project* [Online]. Retrieved March 21, 2017 from <http://venturebeat.com/2014/11/07/h2o-funding/>
38. Cage, D. (2013). *Platforma founder goes in search of big-data answers* [Online]. Retrieved March 21, 2017 from <http://blogs.wsj.com/venturecapital/2013/04/15/platforma-founder-goes-in-search-of-big-data-answers/>
39. Wilson, A. (1999). *ACM Honors Dr. John M. Chambers of bell labs with the 1998 ACM software system award for creating S system* [Online]. Retrieved March 21, 2017 from <http://oldwww.acm.org/announcements/ss99.html>
40. Chambers, J., & Hastie, T. (1991). *Statistical models in S*. Brooks Cole: Wadsworth.
41. Schuster, W. (2014). *Cliff click on in-memory processing, Oxdata H2O, efficient low latency java and GCs* [Online]. Retrieved March 21, 2017 from <https://www.infoq.com/interviews/click-Oxdata>
42. Click, C. (2016). *Winds of change* [Online]. Retrieved March 21, 2017 from <http://www.cliffc.org/blog/2016/02/19/winds-of-change/>
43. H2O. (2016). *H2O* [Online]. Retrieved March 21, 2017 from <http://0xdata.com/about/>
44. Efron, B., & Tibshirani, R. (1994). *An introduction to the bootstrap*. New York: Chapman & Hall/CRC.
45. Hastie, T. J., & Tibshirani, R. J. (1990). *Generalized additive models*. Boca Raton: Chapman & Hall/CRC.
46. Hastie, T., Tibshirani, R., & Friedman, J. H. (2011). *The elements of statistical learning*. New York: Springer.

47. Boyd, S., & Vandenberghe, L. (2004). *Convex optimization* [Online]. Retrieved March 21, 2017 from http://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf
48. Wikipedia. (2017). *H2O (Software)* [Online]. Retrieved March 21, 2017 from [https://en.wikipedia.org/wiki/H2O_\(software\)](https://en.wikipedia.org/wiki/H2O_(software))
49. Oxdata. (2013). *H2O software architecture*. Retrieved March 21, 2017 from http://h2o-release.s3.amazonaws.com/h2o/rel-noether/4/docs-website/developuser/h2o_sw_arch.html
50. Banos, O., Garcia, R., & Saez, A. (2014). *MHEALTH dataset data set. UCI machine learning repository: Center for machine learning and intelligent systems* [Online]. Retrieved February 24, 2017 from <https://archive.ics.uci.edu/ml/datasets/MHEALTH+Dataset>
51. Banos, O., Garcia, R., Holgado, J. A., Damas, M., Pomares, H., Rojas, I., Saez, A., & Villalonga, C. (2014). mHealthDroid: a novel framework for agile development of mobile health applications. In *Proceedings of the 6th international work-conference on ambient assisted living and active ageing (IWAAL)*, Belfast, Northern Ireland.
52. Banos, O., Villalonga, C., Garcia, R., Saez, R., Damas, M., Holgado, J. A., et al. (2015). Design, implementation, and validation of a novel open framework for agile development of mobile health applications. *Biomedical Engineering OnLine*, 14(S2:S6), 1–20.
53. Databricks. (n.d.). *Making machine learning simple: Building machine learning solutions with databricks* [Online]. Retrieved March 21, 2017 from http://cdn2.hubspot.net/hubfs/438089/Landing_pages/ML/Machine-Learning-Solutions-Brief-160129.pdf
54. Kraska, T., Talwalkar, A., Duchi, J., Grith, R., Franklin, M., & Jordan, M. (2013). Distributed machine-learning system. In *Conference on innovative data systems research*.
55. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., et al. (2016). Machine learning in apache spark. *Journal of Machine Learning Research*, 17, 1–7.
56. Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., & Stoica, I. (2014). raphX: Graph processing in a distributed data framework. In *Conference on operating systems design and implementation*.
57. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning Research*, 3, 993–1022.
58. Bradley, J. (2015). *Topic modeling with LDA: MLLib meets GraphX* [Online]. Retrieved March 21, 2017 from <https://databricks.com/blog/2015/03/25/topic-modeling-with-lda-mllib-meets-graphx.html>
59. Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., Stoica, & I. (2013). Discretized streams: Fault-tolerant streaming computing at scale. In *Symposium on operating systems principles*.
60. Freeman, J. (2015). *Introducing streaming k-means in Apache Spark 1.2* [Online]. Retrieved March 21, 2017 from <https://databricks.com/blog/2015/01/28/introducing-streaming-k-means-in-spark-1-2.html>
61. Panda, B., Herbach, J. S., Basu, S., Bayardo, R. J. (2009). Planet: Massively parallel learning of tree ensembles with mapreduce. In *International conference on very large databases*.
62. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine Learning Research*, 12, 2825–2830.
63. Buitink, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., et al. (2013). API design for machine learning software: Experiences from the scikit-learn project. In *European conference on machine learning and principles and practices of knowledge discovery in databases*.
64. Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J. E., et al. (2013). MLI: An API for distributed machine learning. In *International conference on data mining*.
65. Sparks, E. R., Talwalkar, A., Haas, D., Franklin, M. J., Jordan, M. I., & Kraska, T. (2015). Automating model for large scale machine learning. In *Symposium on cloud computing*.
66. Meng, X., Bradley, J., Sparks, E., & Venkataraman, S. (2015). *ML pipelines: A new high-level API for MLLib*. Retrieved March 21, 2017 from <https://databricks.com/blog/2015/01/07/ml-pipelines-a-new-high-level-api-for-mllib.html>
67. Apache Spark. (n.d.). *ML pipelines* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/docs/latest/ml-pipeline.html>

68. Apache Spark. (n.d.). *Extracting, transforming and selecting features* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/docs/latest/ml-features.html>
69. Apache Spark. (n.d.). *Classification and regression* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/docs/latest/ml-classification-regression.html>
70. Apache Spark. (n.d.). *Clustering* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/docs/latest/ml-clustering.html>
71. Apache Spark. (n.d.). *Collaborative filtering* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/docs/latest/ml-collaborative-filtering.html>
72. Apache Spark. (n.d.). *ML tuning: Model selection and hyperparameter tuning* [Online]. Retrieved March 21, 2017 from <http://spark.apache.org/docs/latest/ml-tuning.html>
73. Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1), 1–21.