Big Data Machine Learning using Apache Spark MLlib

Mehdi Assefi[1], Ehsun Behravesh[2], Guangchi Liu[3], and Ahmad P. Tafti[4]

[1]Department of Computer Science, University of Georgia, GA 30602, USA
[2]IEEE Memebr, Kuala Lumpur, Malaysia
[3]Stratifyd Inc. Charlotte, NC 28208, USA
[4]Biomedical Informatics Research Center, Marshfield Clinic Research Institute, WI 54449, USA

***Abstract*—Artificial intelligence, and particularly machine learning, has been used in many ways by the research community to turn a variety of diverse and even heterogeneous data sources into high quality facts and knowledge, providing premier capabilities to accurate pattern discovery. However, applying machine learning strategies on big and complex datasets is computationally expensive, and it consumes a very large amount of logical and physical resources, such as data file space, CPU, and memory. A sophisticated platform for efficient big data analytics is becoming more important these days as the data amount generated in a daily basis exceeds over quintillion bytes. Apache Spark MLlib is one of the most prominent platforms for big data analysis which offers a set of excellent functionalities for different machine learning tasks ranging from regression, classification, and dimension reduction to clustering and rule extraction. In this contribution, we explore, from the computational perspective, the expanding body of the Apache Spark MLlib 2.0 as an open-source, distributed, scalable, and platform independent machine learning library. Specifically, we perform several real world machine learning experiments to examine the qualitative and quantitative attributes of the platform. Furthermore, we highlight current trends in big data machine learning research and provide insights for future work.**

***Index Terms*—Apache Spark MLlib, Big Data Machine Learning, Big Data Analytics, Machine Learning.**

I. Introduction

Digital datasets have been rapidly growing in size and complexity, and the large volume of daily generated data exceeds the boundary of normal processing capabilities, forcing us to take advanced computational infrastructures able to tackle parallel and distributed processing. Efficient mining of such massive data amounts is an extremely challenging practice which requires developing more sophisticated platforms to get extensive big data analysis accurately done in a timely fashion. Big data infrastructures have emerged to address the problem of big data analytics with the use of fast, reliable, and scalable computational architecture, providing excellent quality attributes including elasticity, availability, and resource pooling with the ability of on-demand and ease-of-use self-services [1], [2], [3], [4]. Several big data machine learning frameworks are now available which, aside from lending practical contributions to other scientific disciplines, have demonstrated successful application in healthcare informatics [5], [6], [7], [8], [9], [10], [11], genomic data analysis [12], [13], [14], [15], text mining [16], [17], [18], [19], and stochastic modeling [20], [21], [22] purposes just to name a few.

Apache Spark MLlib is one of the most highly demanded platform independent and open-source libraries for big data machine learning which benefits from distributed architecture and automatic data parallelization. Apache Spark MLlib has been provided with Apache Spark [23], [24], and it offers a set of dominant functionalists for a variety of machine learning tasks, including regression, dimension reduction, classification, clustering, and rule extraction. While machine learning and its effective applications have been studied for a long time in the research community, the study of big data machine learning libraries, such as Apache Spark MLlib has been very limited so far. This contribution is perhaps the first work that leverages a big data machine learning library, the Apache Spark MLlib 2.0, to tackle the problem of big data analytics. An objective of big data analytics is to get advanced computational infrastructures so that large-scale data can be mined and analyzed in a timely and efficient manner. This constitutes the main motivation of the present work. Since big data analytics is computationally intensive, the performance and user experience are impacted by different hardware and/or software configurations. In this paper, we evaluate the impact of different hardware and software configurations with a set of big data analysis tasks. Based on the findings of this study, we provide insights for future big data machine learning-oriented hardware, software, and model design.

The mechanism this paper will discuss is to present, from the computational perspective, the Apache Spark MLlib 2.0 and its capabilities and advantages to big data machine learning. We demonstrate through extensive experiments the benefits of such a highly scalable machine learning library, and highlight the insights that can be drawn by big data analytics. Using several datasets including tens of millions of data records, we demonstrate that we can reliably utilize Apache Spark MLlib for a variety of large-scale machine learning strategies ranging from big data classification to big data clustering and rule extraction. We briefly summarize our main contributions in the following:

- We utilize Apache Spark MLlib 2.0 to initiate a study of big data machine learning on massive datasets including tens of millions of data records. The first and main contribution of the paper is to introduce an exciting but challenging area to the machine learning community. While machine learning and its applications in research and industrial communities have been studied for several years now, the study of big data machine learning has been very limited so far. The present work is expected to bridge the gap between the two areas, opening the doors for different interesting directions from the big data community to the fast-growing machine learning application area.

- We perform several large-scale real world experiments to examine a set of qualitative and quantitative attributes of Apache Spark MLlib 2.0. Furthermore, we establish a comparative study with Weka library (Version 3.7.12) [25] (running on hadoop-2.7 [26]); a very well-known Java-based machine learning library which has been widely used in the community. We evaluate multiple common big data machine learning models, including classification and clustering on real data, and compare their performance on various hardware and software configurations.

The rest of the paper is organized as follows. We begin by introducing Apache Spark MLlib in Section 2. Section 3 explains the materials and methods which consist of a list of Apache Spark MLlib 2.0 components we investigate, plus several large-scale datasets. We then delve into the experimental validations in Section 4. Finally, Section 5 concludes with a summary of our findings and a discussion of several possible directions for future work.

II. Apache Spark MLlib 2.0

Apache Spark [24], [27], [28] as a highly scalable, fast, and in-memory big data processing engine, has been originally developed in the AMPLab at UC Berkeley, and it offers an ability to develop distributed applications using Java, Python, Scala, and R programming languages. It comes with four major libraries, including Apache Spark Streaming, Apache Spark SQL, Apache Spark GraphX, and Apache Spark MLlib [29]. While Apache Spark Streaming as the core scheduling module of Spark, implements stream processing within highly fault tolerant and batch analytics architecture, The Apache Spark SQL implements relational queries to mine different database systems, introducing a data abstraction model called DataFrames [24], [30]. Apache Spark GraphX is a graph processing library on top of the Apache Spark which provides distributed computational models to process two common data structures, such as graph and collection. Apache Spark MLlib is a big data analytics library which provides more than 55 scalable machine learning algorithms that benefit from both data and process parallelization [31]. The library includes implementation of a variety of machine learning strategies, such as classification, clustering, regression, dimension reduction, and rule extraction which enables easy and fast in-practice development of large-scale machine learning applications. Apache Spark MLlib also offers a set of multi-language APIs to evaluate machine learning methods, deploying several computational components that deal with optimization, latent Dirichlet allocation, linear algebra, and feature engineering pipelines [24], [31]. In the recent years, many facets of data science solutions have been amended by such a library [32], [33], [34], [35], [36], [37], and many machine learning scientists and engineers have entered the development of new Apache Spark MLlib components to contribute to the big data analytics community across the world. Figure 1 presents, from the development side, a pathway of the Apache Spark MLlib 2.0 in which the number of unique commits per release has been rapidly growing over the years. Apache Spark MLlib has been in very active development, and at the time of writing this paper, the number of Apache Spark MLlib contributors was above 1000 [38]. Here, we briefly review the recent advances in Apache Spark MLlib applications. Tizghadam *et. al.* [39] proposed an open source, scalable platform called CVST. The system is proposed to be used in smart transport application development. CVST cosists of four major components for resource management, data dissemination, business intelligence, and application. The business intelligence component which is in charge of data analytics, uses MLlib to process the data and deliver it to front-end. Lee *et. al.* [40] proposed an architecture design of academic information system providing services for analyzing students' record patterns. The proposed recommender system uses Apache Spark MLlib to predict and recommend courses for the following semester. SparkText is a text mining framework developed by Ye *et. al.* [41]. The proposed system which employs Apache Spark machine learning and streaming methods along with Cassandra NoSQL database has been executed on a big dataset of medical articles to classify cancer types. Arora [42] analyzed mobile data collected from Web by using K-means algorithm on Apache Spark MLlib. The author offers an effective way of calculating the number of users of the network by clustering based on latitude and longitude values. Lee *et. al.* [43] introduced ALMD - a feature descriptor by considering motion and appearance and employing Apache Spark machine learning library random forest to recognize human activities. An attempt to generate a framework for analyzing the population structure using the next generation sequencing data is made by Hryhorzhevsk *et. al.* [44]. Authors of the paper proposed a distributed computing framework using ADAM combined with MLib, H2O and Apache SystemML. An architecture for automatic machine learning is proposed by Sparks *et. al.* [45]. The system is consisting of resource allocation estimator tunning, and optimizing components. The entire system is built upon Apache Spark and it leverages MLlib and other Spark components. The bigNN which is developed by Tafti *et. al.* [46] is another interesting big data analytics component implemented on top of the Apache Spark, and it is capable to tackle very large-scale biomedical sentence classification. There still exists a list of valuable contributions on big data analytics. Interested readers are referred to [40], [47], [48], [49], [50], [51], [52], [53], [54], [55] for further readings.

III. Materials and Methods

In this section we further explain the materials and methods of the current research study. We shall begin with the Apache Spark MLlib components, and then introduce the datasets.

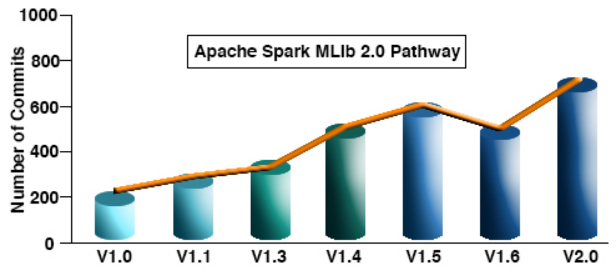


Fig. 1: The development pathway of Apache Spark MLlib 2.0. DataFrame-based APIs in Java and Scala programming languages have been provided by MLlib V 1.2. The MLlib v 1.3 and v 1.5 came with Python and R APIs respectively. Unification of APIs (Dataset &DataFrame), built-in CSV file support, and rapid explorative data analysis offered by Apache Spark MLlib 2.0. It supports for Generalized Linear Models (GLM), Naïve Bayes, Survival Regression, and K-Means in R.

A. Machine Learning Components

To evaluate the ability of the Apache Spark MLlib 2.0 library in analyzing big data sets, we focused on a set of supervised (classification) methods, including SVM (Support Vector Machine), Decision Tree, Naïve Bayes, and Random Forest, along with a widely-used unsupervised (clustering) machine learning algorithm, namely KMeans. To address a comparative study, the machine learning algorithms in Apache Spark MLlib 2.0 library were compared to the same algorithm in Weka library (Version 3.7.12) [25] running on hadoop-2.7 [26]. To work with Apache Spark MLlib 2.0 and also Weka components, We utilized Java2SE 8.1 programing language for all parts of the code and implementations. For each set of the ML algorithms, we employed the same configurations (e.g., regularization, cost, loss, kernel type, seed, etc.) with equal value of parameters offered by both Apache Spark MLlib 2.0 and Weka libraries.

B. Datasets

Six various big datasets were used to analyze and compare the Apache Spark MLlib 2.0 performance. Five datasets from the UCI Machine Learning Repository [56], and a dataset from the US government's Bureau of Transportation Research and Innovative Technology Administration (RITA) [57] Web sites.

The first dataset called "HEPMASS" [58], [59] includes high-energy physics experiments to search for the signatures of exotic particles, and it is associated for binary classification task. The second one named "SUSY" is associated to a binary classification problem to distinguish between a signal process that produces supersymmetric particles and a background process that does not [60], [61].The third dataset called "HIGGS" is related to a binary classification problem to distinguish between a signal process which produces Higgs bosons and a background process which does not [62], [61].The forth dataset called "FLIGHT" [57] which includes flight information from October 1987 to April 2008. There are variables related to flight time, flight origin and destination, flight elapsed

TABLE I: Datasets' attributes

Dataset	Characteristics	Attributes	Data records	Size
HEPMASS	Multivariate	Real	7,000,000	2.20 GB
SUSY	Multivariate	Real	4,999,998	1.81 GB
HIGGS	Multivariate	Real	10,999,998	5.74 GB
FLIGHT	Multivariate	Integer, Nominal	47,113,991	3.09 GB
HETROACT I	Multivariate, Time-Series	Real	13,062,477	879 MB
HETROACT II	Multivariate, Time-Series	Real	13,932,634	977 MB

TABLE II: The employed configurations of two VMs on a VMWARE Cluster environment.

Environment	Number of nodes	HDD	RAM	CPU
ENV1	2	1 TB (in total)	8 GB (each)	4 vCPUs (each)
ENV2	2	1 TB (in total)	16 GB (each)	8 vCPUs (each)

time, arrival airport, delay time, and etc. within the dataset. We utilized this dataset for classification purpose. The fifth and sixth datasets called "HETROACT I" and "HETROACT II" [63], [64]. These heterogeneity datasets for human activity recognition from Smartphone and/or Smartwatch sensors are designed to investigate sensor heterogeneities' impacts on human activity recognition algorithms [64], and we utilized them for clustering purpose. For classification purposes, we employed 75% of every dataset to train the classifier, and 25% to test, all with using 4-fold cross validation to get the Area under ROC (auROC) [65], [66]. Table I illustrates the detailed attributes of each dataset.

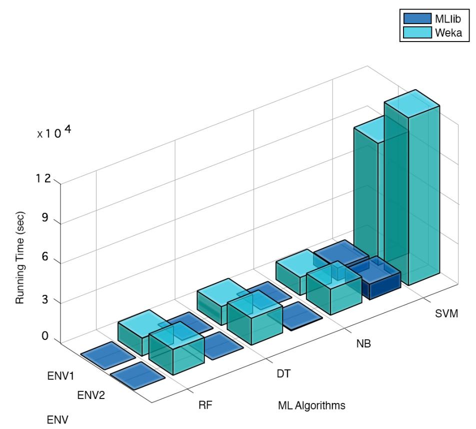
C. Testbed Environments

Two VMs in a VMWARE Cluster environment have been used to obtain experimental results. We utilized two different configurations namely "ENV1" and "ENV2" on both VMs as further illustrated in Table II. The 64-bit CentOS 6.8 operating system with Xeon E5-2690V3 2.6 GHz CPU were used on both VMs.

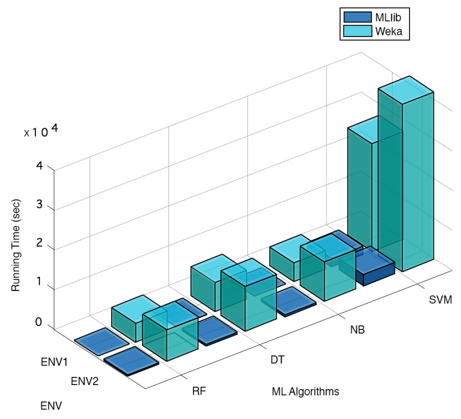
IV. Experimental Results

We focused on the following supervised (classification) methods: SVM (Support Vector Machine), Decision Tree, Naïve Bayes, and Random Forest. We also employed K-Means as an unsupervised (clustering) algorithm to perform experimental studies on both supervised and unsupervised machine learning methods. Here is a summary of our experiments over the six datasets illustrated in Table I. Results present the running time of Mllib and Weka with same hardware setup. The area under ROC(s) obtained by Weka and Apache Spark MLlib with the use of SVM, Decision Tree, Naïve Bayes, and Random Forest methods over four different datasets are presented in Table III. Table IV discusses the experimental results obtained by the K-Mean algorithm across the datasets. Figures 2a, 2b, 2c, and 2d show the running time of 4 selected classification algorithms on the proposed datasets. The running times of K-means on HETROACT I and HETROACT II datasets is summarized in figures 3a and 3b. Summary of the experimental results are:

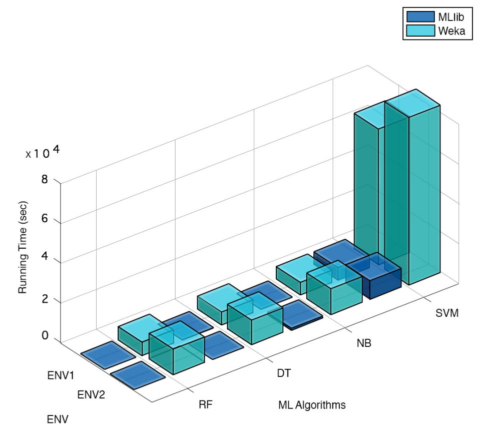
- Area under the ROC obtained by both Weka and Apache Spark MLlib are pretty similar to each other, and the difference is not statistically significant. The small differences between the Area under the ROC achieved by Weka and Apache Spark MLlib might be associated to



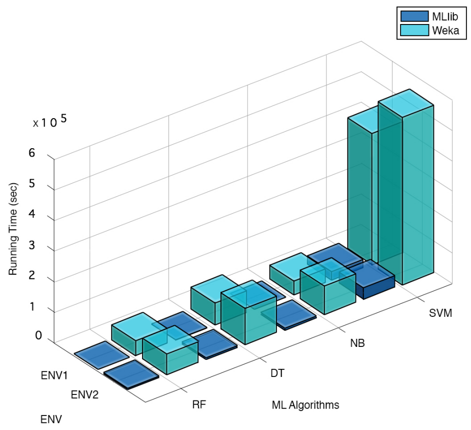
(a) The results obtained on the *Flight* dataset.



(b) The results obtained on the *HEPPMASS* dataset.

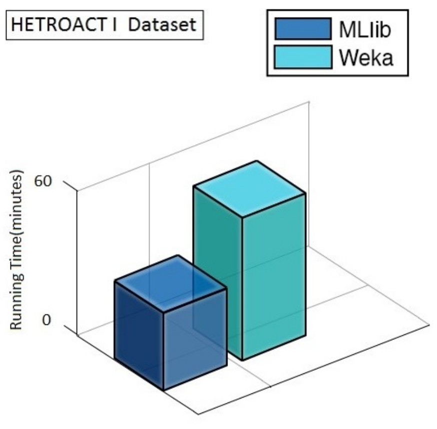


(c) The results obtained on the *HIGGS* dataset.

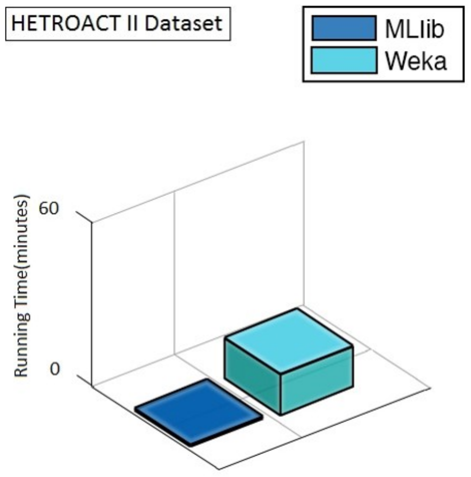


(d) The results obtained on the *SUSY* dataset.

Fig. 2: Running time of Apache Spark MLlib compared to Weka under different classification experiments. NB stands for Naïve Bayes, DT stands for Decision Tree, and RF refers to Random Forest.



(a) Running time of K-Means clustering over the *HETROACT I* dataset.



(b) Running time of K-Means clustering over the *HETROACT II* dataset.

Fig. 3: The running time of Apache Spark MLlib K-Means compared to Weka K-Means clustering component.

TABLE III: Area under ROC associated with the classification algorithms performed by Apache Spark MLlib and Weka library.

Environment	Dataset	Algorithm	MLlib	Weka
	SUSY	SVM	0.7932	0.7829
		Random Forest	0.7614	0.7731
		Decision Tree	0.7528	0.7690
		Nave Bayes	0.7853	0.8031
	HIGGS	SVM	0.5690	0.5543
		Random Forest	0.5680	0.5712
		Decision Tree	0.5829	0.5873
		Nave Bayes	0.5741	0.5722
ENV1	FLIGHT	SVM	0.5189	0.5312
		Random Forest	0.5305	0.5581
		Decision Tree	0.5736	0.5819
		Nave Bayes	0.5570	0.5384
	HEPMASS	SVM	0.9591	0.9543
		Random Forest	0.8948	0.8961
		Decision Tree	0.8995	0.8971
		Nave Bayes	0.9114	0.9398
	SUSY	SVM	0.7932	0.7829
		Random Forest	0.7614	0.7731
		Decision Tree	0.7528	0.7690
		Nave Bayes	0.7853	0.8031
	HIGGS	SVM	0.5690	0.5543
		Random Forest	0.5680	0.5712
		Decision Tree	0.5829	0.5873
		Nave Bayes	0.5741	0.5722
ENV2	FLIGHT	SVM	0.5189	0.5312
		Random Forest	0.5305	0.5581
		Decision Tree	0.5736	0.5819
		Nave Bayes	0.5570	0.5384
	HEPMASS	SVM	0.9591	0.9543
		Random Forest	0.8948	0.8961
		Decision Tree	0.8995	0.8971
		Nave Bayes	0.9114	0.9398

TABLE IV: The general performance of Weka and Apache Spark MLlib K-Means algorithm.

Dataset	MLlib	Weka
HETROACT I	Time: 0 min, 48 sec	Time: 11 min, 40 sec
	Number of Clusters: 5	Number of Clusters: 5
	SSE: 1.2403975185657699E34	SSE: 1.2419981124641322E34
HETROACT II	Time: 19 min, 57 sec	Time: 53 min, 16 sec
	Number of Clusters: 5	Number of Clusters: 5
	SSE: 1.2312227116685513E35	SSE: 1.2619357121584791E35

the detailed parameters of each algorithm in randomly selecting train and test instances while we are doing 4-fold cross validation, or they could come from the very detailed internal parameters of each algorithm.

- The comparative study demonstrates that the Apache Spark MLlib, as expected, is able to be faster in comparison with the Weka components we have utilized, and performing a t-test on the running time matched by either classification algorithms or the clustering method shows statistically significant differences (at $p < 0.01$) between Apache Spark MLlib and Weka.

V. Discussion and Outlook

The amount of digital data being collected is expanding on a daily basis, and as a result, the science of data analytics and processing should be more technologically advanced to enable data scientists to turn this large body of structured and even unstructured data into high quality information and facts. Computer engineers and scientists have already entered the development of big data machine learning components to solve the problem of pattern discovery form large-scale data sources more efficiently, and Apache Spark MLlib is one of the widely-used big data machine learning library available to the community.

Apache Spark MLlib is a very strong tool for big data analytics and as results of our study shows, it presents spectacular performance in terms of running time. Weka, on the other hand, works slower than Apache Spark MLlib under big loads of data samples. Considering different file systems and configurations used by Apache Spark MLlib and Weka, this comparison may not be deemed fair though. We ran MLlib on Spark distributed file system and used Hadoop distributed file system for Weka. But our goal is to show how Spark performs with large sets and Weka is used here as a reliable baseline that is agreed by the research community. There are many features with Weka that Spark can not compete with: (1) There is a big pool of documents and resources available for users, (2) It is very straightforward and easy to use for non-expert users, and (3) It has a perfect graphical user interface. Weka supports a vast variety of machine learning algorithms.

Apache Spark MLlib offers fast, flexible, and scalable implementations of a variety of machine learning components, ranging from ensemble learning and principal component analysis (PCA) to optimization and clustering analysis. Apache Spark MLlib also offer options for distributed processing by parallel processing and support of big data tools that utilize distributed architectures. These criteria will decrease the processing time required and, at the same time, increase the time available to interpret analytics results. This becomes very important when the machine learning task has many predictions to calculate. The distributed architecture can also take advantages of some of the big data tool sets available to help break apart the machine learning component to improve overall running time. Integration is another advantage of Apache Spark MLlib, meaning that the MLlib gains from several software components available in the Spark ecosystem, such as Spark GraphX, Spark SQL, and Spark Streaming, and a wide range of well-organized documentations, including code samples are publicly and freely available to the machine learning community.

Our future work will be focused on adding more practical experiments with most of the Apache Spark MLlib components by utilizing a variety of bigger datasets. As part of our future work, we are working to run an experimental evaluation of Apache Spark MLlib under a diversity of programming languages (e.g., Python and R), clusters and also hardware or/and software configurations, employing a set of big datasets with variety of characteristics within the data.

References

[1] M. Parashar, X. Li, and S. Chandra, *Advanced computational infrastructures for parallel and distributed applications*. John Wiley & Sons, 2010, vol. 66.

[2] D. Talia, "Toward cloud-based big-data analytics," *IEEE Computer Science*, pp. 98–101, 2013.

[3] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: current state and future opportunities," in *Proceedings of the 14th International Conference on Extending Database Technology*. ACM, 2011, pp. 530–533.

[4] A. Abbasi, S. Sarker, and R. Chiang, "Big data research in information systems: Toward an inclusive research agenda," *Journal of the Association for Information Systems*, vol. 17, no. 2, p. 3, 2016.

[5] J. Archenaa and E. M. Anita, "Interactive big data management in healthcare using spark," in *Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC–16)*. Springer, 2016, pp. 265–272.

[6] R. Pita, C. Pinto, P. Melo, M. Silva, M. Barreto, and D. Rasella, "A spark-based workflow for probabilistic record linkage of healthcare data." in *EDBT/ICDT Workshops*, 2015, pp. 17–26.

[7] I. a. Q. M. J. How, D. I. M. H. I. Leave, Y. S. I. F. Question, and T. Y. Why, "A case study: Apache spark."

[8] A. P. Tafti, E. LaRose, J. C. Badger, R. Kleiman, and P. Peissig, "Machine learning-as-a-service and its application to medical informatics," in *International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2017, pp. 206–219.

[9] R. Fang, S. Pouyanfar, Y. Yang, S.-C. Chen, and S. Iyengar, "Computational health informatics in the big data age: a survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, p. 12, 2016.

[10] J. D. Van Horn, "Opinion: Big data biomedicine offers big higher education opportunities," *Proceedings of the National Academy of Sciences*, vol. 113, no. 23, pp. 6322–6324, 2016.

[11] Z. Lv, J. Chirivella, and P. Gagliardo, "Bigdata oriented multimedia mobile health applications," *Journal of medical systems*, vol. 40, no. 5, p. 120, 2016.

[12] M. S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M. J. Okoniewski, "Sparkseq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision," *Bioinformatics*, p. btu343, 2014.

[13] M. Masseroli, P. Pinoli, F. Venco, A. Kaitoua, V. Jalili, F. Palluzzi, H. Muller, and S. Ceri, "Genometric query language: a novel approach to large-scale genomic data management," *Bioinformatics*, vol. 31, no. 12, pp. 1881–1888, 2015.

[14] D. Ding, D. Wu, and F. Yu, "An overview on cloud computing platform spark for human genome mining," in *Mechatronics and Automation (ICMA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2605–2610.

[15] M. Syed, "Using apache spark for scalable gene sequence analysis," Ph.D. dissertation, TEXAS A&M UNIVERSITY-COMMERCE, 2016.

[16] J. Ryan, "Rapidminer for text analytic fundamentals," *Text Mining and Visualization: Case Studies Using Open-Source Tools*, vol. 40, p. 1, 2016.

[17] C. Rong *et al.*, "Using mahout for clustering wikipedia's latest articles: a comparison between k-means and fuzzy c-means in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 565–569.

[18] A. P. Tafti, J. Badger, E. LaRose, E. Shirzadi, A. Mahanke, J. Mayer, Z. Ye, D. Page, and P. Peissig, "Adverse drug event discovery using biomedical literature: a big data neural network adventure," *JMIR Medical Informatics*, 2017.

[19] A. Garcıa-Pablos, M. Cuadros, and G. Rigau, "V3: Unsupervised aspect based sentiment analysis for semeval-2015 task 12," *SemEval-2015*, p. 714, 2015.

[20] H. S. Bhat, R. Madushani, and S. Rawat, "Scalable sde filtering and inference with apache spark,," *Journal of Machine Learning Research W&CP*, 2016.

[21] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska, "Mli: An api for distributed machine learning," in *2013 IEEE 13th International Conference on Data Mining*. IEEE, 2013, pp. 1187–1192.

[22] H. Ji, S. H. Weinberg, M. Li, J. Wang, and Y. Li, "An apache spark implementation of block power method for computing dominant eigenvalues and eigenvectors of large-scale matrices," in *Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conferences on*. IEEE, 2016, pp. 554–559.

[23] "Apache spark," http://spark.apache.org/.

[24] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[25] "Weka library," http://www.cs.waikato.ac.nz/ml/weka/.

[26] "Apache hadoop releases," http://hadoop.apache.org/releases.html.

[27] "Apache spark mllib," http://spark.apache.org/mllib/.

[28] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[29] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[30] M. Frampton, *Mastering Apache Spark*. Packt Publishing Ltd, 2015.

[31] X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *JMLR*, vol. 17, no. 34, pp. 1–7, 2016.

[32] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, "Automating model search for large scale machine learning," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 368–380.

[33] F. Liang, C. Feng, X. Lu, and Z. Xu, "Performance benefits of datampi: a case study with bigdatabench," in *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*. Springer, 2014, pp. 111–123.

[34] M. Armbrust, T. Das, A. Davidson, A. Ghodsi, A. Or, J. Rosen, I. Stoica, P. Wendell, R. Xin, and M. Zaharia, "Scaling spark in the real world: performance and usability," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1840–1843, 2015.

[35] E. R. Sparks, A. Talwalkar, M. J. Franklin, M. I. Jordan, and T. Kraska, "Tupaq: An efficient planner for large-scale predictive analytic queries," *arXiv preprint arXiv:1502.00068*, 2015.

[36] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin, "Large-scale logistic regression and linear support vector machines using spark," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 519–528.

[37] A. G. Shoro and T. R. Soomro, "Big data analysis: Apache spark perspective," *Global Journal of Computer Science and Technology*, vol. 15, no. 1, 2015.

[38] "Github-apache spark," https://github.com/apache/spark/.

[39] A. Tizghadam and A. Leon-Garcia, "Application platform for smart transportation," in *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*. Springer, 2015, pp. 26–32.

[40] M.-S. Lee, E. Kim, C.-S. Nam, and D.-R. Shin, "Design of educational big data application using spark," in *Advanced Communication Technology (ICACT), 2017 19th International Conference on*. IEEE, 2017, pp. 355–357.

[41] Z. Ye, A. P. Tafti, K. Y. He, K. Wang, and M. M. He, "Sparktext: Biomedical text mining on big data framework," *PloS one*, vol. 11, no. 9, p. e0162721, 2016.

[42] S. Arora, "Analyzing mobile phone usage using clustering in spark mllib and pig," *International Journal*, vol. 8, no. 1, 2017.

[43] M. A. Uddin, J. bibi Joolee, A. Alam, and Y.-K. Lee, "Human action recognition using adaptive local motion descriptor in spark," *IEEE Access*, 2017.

[44] A. Hryhorzhevska, M. Wiewiórka, M. Okoniewski, and T. Gambin, "Scalable framework for the analysis of population structure using the next generation sequencing data," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 2017, pp. 471–480.

[45] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, "Automating model search for large scale machine learning," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 368–380.

[46] A. P. Tafti, E. Behravesh, M. Assefi, E. LaRose, J. Badger, J. Mayer, A. Doan, D. Page, and P. Peissig, "bignn: an open-source big data toolkit focused on biomedical sentence classification," in *Proceedings of the IEEE BIG DATA*, 2017.

[47] R. Shyam, B. G. HB, S. Kumar, P. Poornachandran, and K. Soman, "Apache spark a big data analytics platform for smart grid," *Procedia Technology*, vol. 21, pp. 171–178, 2015.

[48] S. Gopalani and R. Arora, "Comparing apache spark and map reduce with performance analysis using k-means," *International Journal of Computer Applications*, vol. 113, no. 1, 2015.

[49] N. Yousefi, M. Georgiopoulos, and G. C. Anagnostopoulos, "Multi-task learning with group-specific feature space sharing," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 120–136.

[50] M. S. Fazli, S. A. Vella, S. N. Moreno, and S. Quinn, "Computational motility tracking of calcium dynamics in toxoplasma gondii," *arXiv preprint arXiv:1708.01871*, 2017.

[51] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "A brief survey of text mining: Classification,

clustering and extraction techniques," *arXiv preprint arXiv:1707.02919*, 2017.

[52] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[53] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health information science and systems*, vol. 2, no. 1, p. 3, 2014.

[54] Z. Lv, H. Song, P. Basanta-Val, A. Steed, and M. Jo, "Next-generation big data analytics: State of the art, challenges, and future research topics," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1891–1899, 2017.

[55] M. Pecaric, K. Boutis, J. Beckstead, and M. Pusic, "A big data and learning analytics approach to process-level feedback in cognitive simulations," *Academic Medicine*, vol. 92, no. 2, pp. 175–184, 2017.

[56] "Uci machine learning repository," http://archive.ics.uci.edu/ml/index.html.

[57] "Us government's bureau of transportation research and innovative technology administration (rita)," http://transtats.bts.gov/DL_SelectFields.asp?Table_ID=236.

[58] "Hepmass dataset," http://archive.ics.uci.edu/ml/datasets/HEPMASS.

[59] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, "Parameterized machine learning for high-energy physics," *arXiv preprint arXiv:1601.07913*, 2016.

[60] "Susy dataset," http://archive.ics.uci.edu/ml/datasets/SUSY.

[61] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, 2014.

[62] "Higgs dataset," http://archive.ics.uci.edu/ml/datasets/HIGGS.

[63] "Heterogeneity activity recognition dataset," http://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition.

[64] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen, "Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 127–140.

[65] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[66] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[67] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics." in *NSDI*, 2016, pp. 363–378.

[68] S. Ma and Z. Liang, "Design and implementation of smart city big data processing platform based on distributed architecture," in *Intelligent Systems and Knowledge Engineering (ISKE), 2015 10th International Conference on*. IEEE, 2015, pp. 428–433.

[69] "Spark standalone mode," https://spark.apache.org/docs/2.1.0/spark-standalone.html.

APPENDIX A
HOW TO RUN APACHE SPARK ON A CLUSTER

The $SparkContext$ object in the main program, which is also called the driver program, is utilized to run Apache Spark applications on a cluster environment.

```
public static void main(String[] args) {
 SparkConf sparkConf =
 new SparkConf().setAppName("JavaSample");
 sparkConf.setMaster("local[*]");
 JavaSparkContext jsc =
 new JavaSparkContext(sparkConf);
```

The $SparkContext$ is able to connect to a variety of cluster managers, such as YARN and Mesos, which manage in allocating resources across different applications. Once connected, Apache Spark procures executors on available nodes in the cluster environment. It then sends the application code (e.g., JAR files) to the executors, and finally, the $SparkContext$ sends tasks to the executors to run.

Spark Standalone Mode

Apache Spark also provides an easy-to-use standalone deployment mode, in which we can easily put a compiled version of Apache Spark on every single node on the cluster environment [69]. In this Appendix, and for example, we are using only two nodes, a master node and a slave node, assuming to employ Java programming language. In doing so, we have to do the following steps respectively:

(1) Install Java on both nodes.
(2) Set JAVA_HOME environmental variable on both nodes.
(3) Make sure both nodes can ping each other by host-name.

We can then start a standalone master server by:

```
./sbin/start-master.sh
```

The master node should print a spark://HOST:PORT URL in a way we can use that to connect workers to the master node, or pass as the $master$ argument to the $SparkContext$. We are able to start one or more worker nodes and connect them to the $master$ using:

```
./sbin/start-slave.sh <master-spark-URL>
```

Further information is available at [69]. Here, we also present a sample code to see how we can use an Apache Spark MLlib's SVM and Decision Tree classification components:

```
// SVM
String path =
Resource.getPath("data/HEPMASS.txt");
JavaRDD<LabeledPoint> data =
MLUtils.loadLibSVMFile(jsc.sc(), path)
.toJavaRDD();
JavaRDD<LabeledPoint> training =
data.sample(false, 0.75, 11L);
training.cache();
JavaRDD<LabeledPoint> test =
data.subtract(training);
int numIterations = 120;
final SVMModel modelSVM =
SVMWithSGD.train(training.rdd(),
numIterations);
// Decision Tree
JavaRDD<LabeledPoint>[] splits =
data.randomSplit(new double[]{0.75,0.25});
JavaRDD<LabeledPoint> trainingData =
splits[0];
JavaRDD<LabeledPoint> testData =
splits[1];
Integer numClasses = 3;
Map<Integer, Integer> categoricalFeaturesInfo =
new HashMap<>();
String impurity = "gini";
Integer maxDepth = 5;
Integer maxBins = 32;

final DecisionTreeModel modelDT =
DecisionTree.trainClassifier(trainingData,
numClasses,categoricalFeaturesInfo, impurity,
maxDepth, maxBins);
```