Big data and machine learning framework for clouds and its usage for text classification

István Pintye, Eszter Kail, Péter Kacsuk
Institute for Computer Science and Control
Hungarian Academy of Sciences
Budapest, Hungary
pintye.istvan@sztaki.mta.hu

Péter Kacsuk
University of Westminster
London, UK
P.Kacsuk@westminster.ac.uk

***Abstract*— The paper describes a big data and AI application development and execution framework that was originally developed for MTA Cloud (an OpenStack based cloud) but could be used on other clouds including Amazon, OpenStack, OpenNebula and CloudSigma. The paper explains the concept and components of the big data and AI environment and illustrates its usage by a text classification application.**

Keywords—machine learning; big data; parallel and distributed execution; cloud;

I. INTRODUCTION

Researches in different scientific fields (Natural Sciences, Physics, Political Science) often require huge computational resources and storage capacity to handle real Big Data. Traditional sequential data processing algorithms are not sufficient to analyze this large volume of data. For efficient processing and analysis new approaches, techniques and tools are needed.

Moreover, cloud infrastructures and services are becoming even more popular and are playing an appropriate and widely used role to address the computation need of many scientific and commercial Big Data applications. Their widespread usage is a consequence of the dynamic and scalable nature of the services provided by cloud providers.

However, the data scientists face several problems once they start planning the use or deployment of any Big Data platform on cloud(s). On one hand, the selection of the appropriate cloud provider(s) is always a cumbersome process since the potential user community has to take into consideration several factors and trade-offs even if they need only a generic Infrastructure-as-a-Service (IaaS) provider: private institutional (e.g. SZTAKI Cloud [1], federated cloud (e.g. MTA Cloud [2] or pan-European EGI FedCloud [3]) or public cloud (e.g. Amazon [4]).

The Hungarian Academy of Sciences (MTA) provides free IaaS cloud (MTA cloud) services for research communities and easy to use, dynamic infrastructures adapted to the actual project requirements. MTA Cloud was established to accelerate research for the scientists of MTA. Nearly 100 projects have been run on MTA Cloud since its opening and more and more projects require to use Big Data and machine learning applications. However, the large number of AI tools available for clouds are very complex and their proper deployment and configuration requires significant learning of both the tools and the underlying cloud. Furthermore, tools supporting different layers like user interface layer, language layer, machine learning layer, deep learning layer are not always compatible and hence it requires further skill to select the right tools from each layer in a way that they should be able to work together in an AI environment.

Recognizing this problem, we have decided to develop so-called AI reference architectures that can support the solution of certain AI application classes and can run in the cloud in a reliable and robust way and can easily be deployed and used by the end-user scientists. The ultimate goal is to develop a large set of AI reference architectures for a large set of various AI problem classes.

The AI reference architectures have been created in three steps:

1. Development and publication of a cloud orchestrator called Occopus that enables the fast creation of complex application frameworks in the cloud based on Occopus infrastructure descriptors even by novice cloud users.
2. Development and publication of the Occopus infrastructure descriptors for generic AI reference architectures like for example: Jupyter, Python, Spark ml, Spark cluster and HDFS.
3. Development and publication of application-oriented environments for various AI application domains.

To demonstrate the third step, we use a text classification application provided by the POLTEXT (Text Mining of Political and Legal Texts) Incubator Project of MTA Centre for Social Sciences. This problem is complex enough to demonstrate the advantages of using the framework we have created for supporting big data and AI applications.

The structure of the paper is as follows. The next section introduces the IaaS MTA cloud and its major services to create the big data and AI development and execution framework. Section III. introduces our text-classification example with detailed stepwise specification. Section IV. summarizes the lessons learned from this real use case performed on MTA community cloud.

II. Components and services of the big data and AI framework

A. MTA cloud and Occopus

MTA Cloud was founded in 2015, when the Wigner Data Center and the Institute for Computer Science and Control (MTA SZTAKI) collaborated to establish a community Cloud for the member institutes of the Hungarian Academy of Sciences. MTA Cloud has currently more than 80 active projects from over 20 research institutes including among others the Institute for Nuclear Research, the Research Centre for Astronomy and Earth Sciences and other academic and research institutes.

In order to raise the abstraction level of the IaaS MTA Cloud we have developed Occopus a cloud orchestrator and manager tool by which complex infrastructures like Hadoop or Spark clusters can easily be built based on predeveloped and published Occopus infrastructure descriptors. The Occopus cloud orchestrator can be deployed in MTA Cloud by any user and once Occopus is deployed it can be used to build the selected infrastructure (e.g. Spark cluster) in MTA Cloud. A tutorial explaining the deployment of Occopus is available on the web page of MTA Cloud (in Hungarian) [2]. The novelty of Occopus was described and compared with similar cloud orchestrators in [6]. Here we mention only one of its main advantages. Its plugin architecture enables the use of plugins for various cloud systems and hence AI reference architectures created by Occopus are easily portable among various cloud systems like Amazon, Azure, OpenStack, OpenNebula and CloudSigma.

B. Support for parallel data storage and processing – Apache Hadoop

Apache Hadoop is an open source software platform for distributed storage and processing of very large data sets on computer clusters. Due to the special storage method, which is based on a distributed file system (HDFS, Hadoop Distributed File System [7]) Hadoop can process efficiently terabytes of data in just minutes, and even petabytes in hours.

HDFS uses the MapReduce [8] paradigm that was proposed by Google and found wide-spread popularity. HDFS has a master/slave architecture. It means that the nodes apart from the Client machine are Master nodes and Slave nodes. Master node supervises the mechanism of data storing in HDFS and running parallel computations (Map Reduce) on all that data. An HDFS cluster consists of a single NameNode, a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. The NameNode oversees and coordinates the data storage function. Internally, a file is split into one or more blocks, which are stored in a set of DataNodes. NameNode provides a map of where the data blocks are in the cluster. JobTracker oversees and coordinates the parallel processing of data using MapReduce. Slave Nodes make up the vast majority of machines, they store the data and run the computations. Each slave runs both a DataNode and a TaskTracker daemon that communicate with and receive instructions from their Master nodes.

The Occopus infrastructure descriptors for such a Hadoop/HDFS cluster have been developed in SZTAKI and are published on the web page of MTA Cloud [2] as well as on the web page of Occopus [9].

C. Support for high performance, distributed data processing -Apache Spark

Apache Spark [10] is an open source, fast and general-purpose cluster framework, designed to run high performance data analysis applications. Instead of the Apache Hadoop's Map Reduce programming paradigm [8], it performs internal computational data processing that results in a more flexible and faster run. The module uses a parallel data processing framework that stores data in memory and, if necessary, on disk. This type of approach exceeds up to ten times the speed of Hadoop Map Reduce data processing [8].

Apache Spark was written in Scala, and the most important of its favorite features are its highly developed easy-to-use APIs, such as Scala, Java, Python and R, designed specifically for handling large data sets. From an engineering perspective these APIs provide the biggest advantages and reason why choosing the Spark framework. In addition to the Spark Core API, there are other libraries in the Spark Ecosystem, providing additional opportunities for large data analysis and machine learning. These include Spark SQL for structured data processing, MLlib [11] for Machine Learning, etc.

It is important to emphasize that Apache Spark is not a substitute for Apache Hadoop, but a kind of extension of it. Spark has been designed to be able to read and write data from Hadoop's own distributed file system (HDFS), and other storage systems such as HBase or Amazon S3.

D. Spark Machine learning library

Apache Spark MLlib [11] is the Apache Spark machine learning library consisting of common learning algorithms and utilities. As the core component library in Apache Spark, MLlib offers numerous supervised and unsupervised learning algorithms, from Logistic Regression to k-means and clustering, collaborative filtering, dimensionality reduction, and underlying optimization primitives.

As the next step of building a Big Data and AI oriented environment for MTA Cloud users we have developed the Occopus infrastructure descriptors for Spark/HDFS clusters and published them both on the web page of MTA Cloud [2] and on the web page of Occopus [9]

E. Interactive Development Environments

With the above-mentioned frameworks big data and machine learning algorithms can easily be executed in a parallel manner. In order to support scientists from different research fields we also support interactive development environments that are easy to use with various programming languages and are very popular among the research communities.

RStudio [12] is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and work space management. RStudio Desktop is a standalone desktop application that in no way requires or connects to the RStudio Server.

RStudio Web Server is a Linux server application that provides a web browser/based interface to the version of R running on the server. Deploying R and RStudio on a server has a number of benefits: the ability to access R workspace from any computer at any location; sharing of code, data, and other files with colleagues; allowing multiple users to share access to the more powerful computing resources available on a server; control access to data in a centralized manner; centralized installation and configuration of R, R packages and other libraries.

Jupyter Notebooks [13] are starting to become extremely popular especially in education and field of empirical research. The reason for Jupyter's great success stems from the clear advantages of literate programming and improved web browser technologies. Literate programming is a software development style pioneered by Stanford computer scientist, Donald Knuth. Literate programming allows users to formulate and describe their thoughts with prose, supplemented by mathematical equations, as they prepare to write code blocks. It excels at demonstration, research, and teaching objectives especially for science.

There are a lot of free and open source Jupyter Notebook codes on numerous topics in many scientific disciplines, such as machine learning, social sciences, physics, computer science, etc. They have LaTeX support for mathematical equations with MathJax, a web browser enhancement for display of mathematics. These notebooks can be saved and easily shared in ipynb JSON format. They can also be committed to version control repositories such as git and the code sharing site github.

Jupyter notebooks can be viewed with nbviewer technology which is supported by github. Moreover, because these notebook environments are for writing and developing code, they offer many niceties available in typical Interactive Development Environments (IDEs) such as code completion and easy access to help.

As part of the second step of providing generic big data and AI platforms for scientists we have extended the Spark/HDFS cluster with both RStudio Web Server and Jupyter Notebook and created the necessary Occopus infrastructure descriptors. As a result, two types of Spark-oriented reference architecture can be deployed by Occopus on MTA Cloud depending on the actual needs of the users:

1. RStudio Web Server, Spark, HDFS for R users
2. Jupyter Notebook, Spark, HDFS for Python, Scala and Java (from version 9) users

These reference architectures are the starting points for the actual big data or AI applications.

III. TEXT CLASSIFICATION SCENARIO

The third step was the usage of the developed reference architectures for various big data and AI application domains. In this paper we have selected the text classification domain to illustrate the usage of the Spark-oriented reference architecture.

MTA Centre for Social Sciences wanted to solve the following problem on MTA Cloud: The coding of public policy major topics on various legal and media corpora serves as an important input for testing a wide range of hypotheses and models in political science. This fundamental work has till recently mostly been conducted by double-blind human coding, which is still considered the gold-standard for categorizing text in this field. This method, however, is both rather expensive and increasingly unfeasible with the growing size of available corpora. Different forms of automated coding, such as dictionary-based and supervised learning methods, offer a solution to these problems. But these methods are themselves also reliant on appropriate dictionaries and/or training sets, which need to be compiled and developed first.

We have provided the architecture for them described in Section II/E and at the same time demonstrated for them how to use this architecture for solving their problem. After the demonstration they started to use the RStudio version of the framework meanwhile we have also investigated possible solutions for the problem using the Jupyter Notebook version. Here we show our approach to solve the problem. The steps of solving the above described text classification problem are shown in Figure 1. This simple figure in fact, represents several different execution pipelines depending on the choice of the user. With the use of the Jupyter Notebook, Spark, HDFS architecture we were able to execute and evaluate the different classification pipelines in parallel. In the next paragraphs the different stages of our Spark-based pipelines are detailed.

1) Disribute the data

The first stage is to upload the data (text) into the HDFS system in an appropriate form. At first a Resilient Distributed Dataset is built, which is the basic data structure of Spark by dividing the dataset into logical partitions. These partitions may be computed in parallel on different nodes of the cluster.

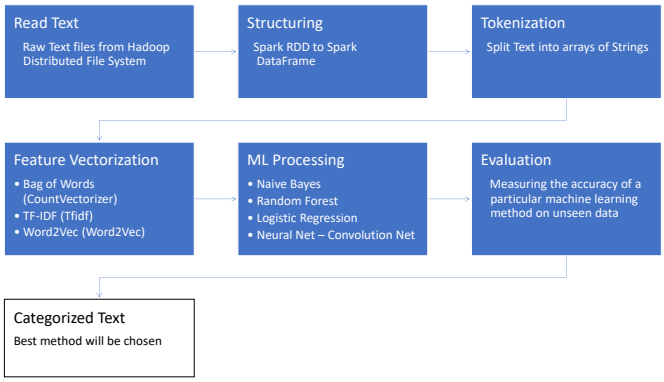

Figure 1 Text processing pipeline

2) Data structuring

The second stage is the data structuring step. Apache Spark SQL is a module for structured data processing in Spark. Spark SQL module supports operating on a variety of data sources through the DataFrame API. DataFrame is a distributed collection of data organized into named columns. Actually, it is equal to the table concept in relational database systems or a dataframe in R/Python. DataFrame contains rows with Schema. It can scale from kilobytes of data on the single laptop to petabytes of data on a large cluster. A DataFrame can be operated on using relational transformations such as filter, select, group by, sort, etc. Like Apache Spark in general, Spark SQL in particular is all about distributed in-memory computations on scale.

3) Text pre-processing

The stored and structured data should be transformed into an appropriate input form for the machine learning algorithm (e.g.: neural networks). This is called text pre-processing. The next stage is therefore the text pre-processing which can have several sub-steps including tokenization, stop-word, stemming. We restricted our pipeline to use only the tokenization sub-step.

Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. For example, in the text string of a sentence the raw input (series of characters) must be explicitly split into tokens with a given space delimiter in the same way as a natural language speaker would do. Spark machine learning library (mllib) has a lot of built in functions for text mining such as RegexTokenizer. Therefore, users of the Spark environment shown in Figure 1 do not have to develop any new software for tokenization, just use the Spark ML RegexTokenizer function.

4) Feature vectorization

Features in our text-classification problems mean to find words, or terms that can represent some special characteristics of the input text. Of course, this feature should be represented in a form of a vector. Accordingly, the next stage in our pipeline of Fig. 1 is feature vectorization. There are different kinds of feature vectorization algorithms and many of them are supported by the SparkML library. In the next paragraphs the applied feature vectorization and word embedding methods are briefly introduced.

a) Bag-of-Words

The bag-of-words (BOW) algorithm provides feature extraction capabilities. As the name suggests, it does not keep the words structured just a "bag" of words. It gives back a histogram of the words within the text, i.e., considering each word count as a feature. The algorithm consists of two phases: first it builds a vocabulary of the known words and then it measures the presence of these words in the different documents related to the corpora.

CountVectorizer function of Spark ML implements this concept by converting a collection of text documents to vectors of token counts. It can be used to extract the vocabulary and to generate an array of strings from the document.

b) TF-IDF

Term frequency-inverse document frequency (TF-IDF) is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus. Terms with high frequency within a document have high weights. In addition, terms frequently appearing in all documents of the document corpus have lower weights. TF-IDF has been traditionally applied in information retrieval systems, because it is capable highlight documents that are closely related to a term but not to an exact string-match. Spark ML function that supports this method is IDF.

c) Word2Vec

Bag-of-Words and TF-IDF hold no information about the meaning of the word, how it is used in language and what is its usual context (i.e. what other words it generally appears close to). Word embeddings try to "compress" large one-hot word vectors into much smaller vectors (a few hundred elements) which preserve some of the meaning and context of the word.

Word2Vec is a sophisticated word embedding technique, which is based on the idea that words that occur in the same contexts tend to have similar meanings. The training objective of Word2Vec is to learn word representations that can predict its context in the same sentence or in the given corpus. This model maps each word to a unique and fixed-size vector that can be used as features for document similarity calculations and classification respectively.

The context of the word is the key measure of meaning that is utilized in Word2Vec. Words which have similar contexts share meaning under Word2Vec, and their reduced vector representations will be similar. The built-in word2vec

algorithm uses the skip-gram neural network model. In the skip-gram model version of Word2Vec, the goal is to take a target word i.e. "sat" and predict the surrounding context words. This involves an iterative learning process, that was performed by a neural network with one hidden layer consisting of 300 neurons.

The end product of this learning will be an embedding layer in a network – this embedding layer is a kind of lookup table – the rows are vector representations of each word in our vocabulary.

5) ML methods in text classification - supervised learning

In this phase of the work we use the different built-in machine learning algorithms, that are shortly introduced in the next paragraphs.

a) Random Forest

Random forests are ensembles of decision trees [14]. Decision trees and their ensembles are very popular methods classification and regression type tasks, since they are easy to interpret, handle categorical features, can be extended to the multiclass classification setting, and are able to capture non-linearities and feature interactions.
The spark.ml implementation supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features. The implementation partitions data by rows, allowing distributed training with millions or even billions of instances.
In spark.ml Decision Tree classifier is available via the DecisionTreeClassifier() method [15].

Random forests combine many decision trees in order to reduce the risk of overfitting. Random forests train a set of decision trees separately, so the training can be done in parallel. The algorithm injects randomness into the training process so that each decision tree is a bit different. Combining the predictions from each tree reduces the variance of the predictions, improving the performance on test data.
In spark.ml implementation random forests is available via RandomForest() method [16].

b) Naïve Bayes

"Bayes" is named from the famous Bayes' Theorem in probability, and "Naive" is because of the strong (naive) independence assumptions between every pair of features.
A feature's value is the frequency of the word (in multinomial Naive Bayes) or a zero or one indicating whether the word was found in the document. Naïve Bayes method in Spark computes the conditional probability distribution of each feature given each label. It applies Bayes' theorem to compute the conditional probability distribution of each label given an observation [15].

c) Multinomial logistic regression

In terms of its structure, logistic regression can be thought as a neural network with no hidden layer, and just one output node. Instead of fitting a straight line or hyperplane, the logistic regression model uses the logistic function to squeeze the output of a linear equation between 0 and 1. In our case the number of inputs were equal with the number of words coming from the bag of words, the tf-idf model [15].

d) Multi Layer Perceptron (Neural Network)

We have aggregated the word vectors of each word in a document, calculating mean to get one vector representation of each document. Now each document is represented by a vector with 300 dimensions. The values of the vectors are the inputs of or fully connected neural network (or feedforward artificial neural network).
Neural net consists of multiple layers of nodes. The layers are fully connected to the next layer in the network.
The input layer represents the input data. All other nodes map inputs to outputs by a linear combination of the inputs with the node's weights w and bias b after then applying an activation function. In spark the nodes in intermediate layers use sigmoid (logistic) function, and this property is not changeable. The last nodes in the output layer use softmax function where the number of nodes in the output layer corresponds to the number of classes.

e) Convolutional Nerual Net (CNN)

In order to feed data to a CNN, we have to ensure that each word vector is fed to the model in a sequence that matches the original document. The dimension of the vector we have for the whole document is the length (or the number of words in the document) times the dimension of the vector (in our case 300) which represents the current word. It is important to note that each word has a fix and same length of vector representation.

A neural network model will expect all the data to have the same dimension, but in case of different documents, they have different lengths. This can be handled with padding.
By padding the inputs, we decide the maximum length of words in a document, then zero pads the rest, if the input length is shorter than the designated length. In the case where it exceeds the maximum length, then it also truncates either from the beginning or from the end. In other words, each document is represented as a matrix, where rows are the words and the columns are the Word2Vec features. This transormation enables our data to be fed into a Convolutional Neural Net (CNN).

6) Evaluation

The final stage is testing, measuring, evaluating and ranking of the classification models and then to choose the best algorithm to classify the new incoming document.

In our experiment we were able to combine all the feature vectorization methods with all machine learning algorithms (shown in Fig. 1) with 2 exceptions:
A) In Naïve Bayes method feature values must be non-negative while the Word2Vec method produces real numbers.
B) Convolution neural net as a classifier can handle data which have the same size of dimensions. As we discussed earlier only the word2vec method can produce a proper input for convolutional net, the bag-of-word, and tf-idf methods cannot.
All the experiments were conducted on eleven-node-cluster, with one master and ten worker nodes. Each node has 8 virtual CPU cores and 16GB of RAM. The overall computing capacity consisted of 80 virtual CPUs and 160GB of RAM. We found that the Word2Vec feature combined with Convolutional Neural Net machine learning algorithm gave the best performance.

IV. Conclusion

We have developed a big data and AI application development and execution framework that needs three major steps to be created:

1. Occopus to define and deploy the required infrastructure in the target cloud. This was created by SZTAKI.
2. Occopus infrastructure descriptors for the generic big data and AI tools and environments like Hadoop, HDFS, Spark, Jupyter Notebook, RStudio Web Server. These are provided as AI reference architectures developed by SZTAKI and can be used in according to the actual AI application class.
3. A concrete application-oriented big data and AI application development and execution framework that is built by Occopus according to the Occopus infrastructure descriptors that are selected, customized and parameterized by the user. The customization and parameterization process is described in detail in the tutorials on the reference architectures provided at the Occopus web page.

In this paper we have demonstrated how to use a big data and AI application development and execution reference architecture tailored for text classification applications. Due to the fast creation of the required Spark environment and the available resources in MTA Cloud we were able to try and test all the possible text classification pipelines that are presented in Fig 1.
Although the presented big data and AI application development and execution framework was created and tested on MTA Cloud it can be used on other clouds including Amazon, Azure, OpenStack, OpenNebula and CloudSigma due to the plugin architecture of the underlying Occopus cloud orchestrator [6]. Many components of the described AI reference architectures are available on the Occopus web page as executable tutorials and the following two Spark-oriented reference architectures are available at the MTA Cloud web page as tutorials:

1. RStudio Web Server, Spark, HDFS for R users
2. Jupyter Notebook, Spark, HDFS for Python, Scala and Java (from version 9) users

Future work will intend to create further AI reference architectures to cover further AI application classes.

Acknowledgment

We thank for the usage of MTA Cloud (https://cloud.mta.hu) that significantly helped us achieve the results published in this paper. We would also like to acknowledge the support of the Text Mining of Political and Legal Texts (POLTEXT) Incubator Project, MTA Centre for Social Sciences.

References

[1] "SZTAKI Cloud home - SZTAKI Cloud." [Online]. Available: https://cloud.sztaki.hu/en/home. [Accessed: 01-Apr-2019].

[2] "MTA Cloud | MTA Cloud." [Online]. Available: https://cloud.mta.hu/. [Accessed: 01-Apr-2019].

[3] E. Fernández-del-Castillo, D. Scardaci, and Á. L. García, "The EGI Federated Cloud e-Infrastructure," Procedia Comput. Sci., vol. 68, pp. 196–205, Jan. 2015.

[4] "Whitepapers – Amazon Web Services (AWS)." [Online]. Available: https://aws.amazon.com/whitepapers/. [Accessed: 01-Apr-2019].

[5] "Laboratory of Parallel and Distributed Systems | MTA SZTAKI." [Online]. Available: https://www.sztaki.hu/en/science/departments/lpds. [Accessed: 01-Apr-2019].

[6] J. Kovács and P. Kacsuk, "Occopus: a Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures," J. Grid Comput., vol. 16, no. 1, pp. 19–37, Mar. 2018.

[7] "HDFS Architecture Guide." [Online]. Available: https://hadoop.apache.org/docs/current1/hdfs_design.html. [Accessed: 01-Apr-2019].

[8] "MapReduce Tutorial." [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. [Accessed: 01-Apr-2019].

[9] "Welcome - Occopus." [Online]. Available: http://occopus.lpds.sztaki.hu/de/. [Accessed: 01-Apr-2019].

[10] "Apache SparkTM - Unified Analytics Engine for Big Data." [Online]. Available: https://spark.apache.org/. [Accessed: 01-Apr-2019].

[11] "MLlib | Apache Spark." [Online]. Available: https://spark.apache.org/mllib/. [Accessed: 01-Apr-2019].

[12] "Open source and enterprise-ready professional software for data science - RStudio." [Online]. Available: https://www.rstudio.com/. [Accessed: 01-Apr-2019].

[13] "The Jupyter Notebook — IPython." [Online]. Available: https://ipython.org/notebook.html. [Accessed: 01-Apr-2019].

[14] L. Breiman, "Random Forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[15] "Classification and regression - MLlib main guide." [Online]. Available: https://spark.apache.org/docs/latest/ml-classification-regression.html.

[16] "Ensembles - RDD-based API - Spark 2.4.0 Documentation." [Online]. Available: https://spark.apache.org/docs/latest/mllib-ensembles.html. [Accessed: 01-Apr-2019].