Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

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

István Pintye[1] | Eszter Kail *[1] | Péter Kacsuk[1,2] | Róbert Lovas[1,3]

[1]Institute for Computer Science and Control (SZTAKI), Budapest, Hungary
[2]University of Westminster, London, UK
[3]Óbuda University, Budapest, Hungary

Correspondence
Eszter Kail Email: eszter.kail@sztaki.hu

Present Address
SZTAKI, Kende u. 13-17, Budapest H-1111, Hungary

Summary

Reference architectures for big data and machine learning include not only interconnected building blocks but important considerations (among others) for scalability, manageability and usability issues as well. Leveraging on such reference architectures, the automated deployment of distributed toolsets and frameworks on various clouds is still challenging due to the diversity of technologies and protocols. The paper focuses particularly on the widespread Apache Spark cluster with Jupyter as the particularly addressed framework, and the Occopus cloud-agnostic orchestrator tool for automating its deployment and maintenance stages. The presented approach has been demonstrated and validated with a new, promising text classification application on the Hungarian academic research infrastructure, the OpenStack-based MTA Cloud. The paper explains the concept, the applied components, and illustrates their usage with real use-case measurements.

KEYWORDS:
machine learning, big data, parallel and distributed execution, cloud, reference architectures, text classification

1 | INTRODUCTION

Research in different scientific fields (e.g. natural and social sciences) often require extremely huge computational resources and storage capacity to handle Big Data problems. 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 necessary.

Moreover, cloud infrastructures and services are becoming even more popular and are nowadays widely used to address the computation and storage requirements of many scientific and commercial Big Data applications. Their widespread usage is a consequence of the dynamic and scalable nature of the services maintained by cloud providers.

However, there are several challenges that a data scientist has to face when planning the use or deployment of any Big Data platform on cloud(s)[1]. The selection of the appropriate cloud provider(s) is always a tiresome process since several factors has to be considered, even when only a generic Infrastructure-as-a-Service (IaaS) provider is required: private (e.g. Agrodat Cloud[2]), federated (e.g. MTA Cloud[3] or pan-European EGI FedCloud[4]), or public cloud (e.g. Amazon AWS[5]).

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 deployed 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 artificial intelligence (AI) tools available for clouds are very complex, and their proper deployment and configuration require deep understanding of both the

tools and the underlying cloud infrastructure. Furthermore, tools supporting different layers, the user interface layer, language layer, machine learning layer, deep learning layer are not always compatible; hence it requires engineering skills to apply and configure the right tools from each layer in a compatible way, so that they should be able to work together in an AI environment.

After recognizing this problem, we have decided to develop so-called AI reference architectures to support the development of certain AI application classes which could be implemented 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. These AI reference architectures have been created in three steps:

1. Creation of Occopus cloud orchestrator[6] which enables the fast creation of complex application frameworks in the cloud. This method is facilitated by using infrastructure descriptors; therefore, even novice cloud users are able to work with it.

2. Development and publication of the Occopus infrastructure descriptors for generic AI reference architectures[1] including 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.

Our main contribution is that our work gives a general and reproducible approach for building reference architectures to support various machine learning-based research studies, easily without any special IT experiences or knowledge. While in the literature, there exist several solutions for text-classification problems based on Apache Spark[7,8]; our work is not limited to text classification scenarios or particular use cases. It gives the possibility to be used by several scientific communities. The environment is also not limited to specific cloud environments since it can be applied almost in every cloud. Moreover, the deployed infrastructure is reproducible based on the descriptors available at the official website of the Hungarian Scientific Cloud[9].

The presented work significantly extends the achievements presented at the *International Workshop on Science Gateways (IWSG 2019)*[10]. In this paper we put our previous approach and solution into a much wider context and validated the results in cloud environment. Therefore, we added the reference architecture concept and necessarily extended the related works (see Section 2), moreover, we significantly improved the description about the application with more details on our measurements, experiments, and findings (see Section 4).

The structure of the paper is as follows. The next section outlines the main motivations and background concerning reference architectures with special focus on AI. Section 3 introduces the IaaS MTA cloud and its major services to create the big data and AI development and execution framework leveraging on reference architectures. Section 4 introduces our text-classification example with detailed stepwise specification. Section 5 summarizes the lessons learned from this real use case performed on MTA community cloud.

2 | RELATED WORKS ON REFERENCE ARCHITECTURES FOR MACHINE LEARNING

Machine Learning-as-a-service (MLaaS) on cloud supports various fundamental steps for machine learning (ML), such as data pre-processing, model training, and model evaluation, with further prediction. Feeding the model with data and using the prediction results can be bridged with on-premise IT infrastructures through APIs. Amazon SageMaker, Microsoft Azure Machine Learning Studio, and Google Cloud AI hub / platform are three leading commercial clouds offering MLaaS. For the various AI application scenarios, the major cloud providers offer reference architectures (such as Azure Reference Architectures[11]) including recommended practices, along with considerations for scalability, availability, manageability, and security. Similar state-of-the-art reference architectures are available from several HPC vendors in order to broaden the landscape: Hewlett Packard Enterprise elaborated its reference architecture for AI[12] (with TensorFlow and Spark) that copes not only with open-source software components, but their own propriety cluster-based hardware platform as well. IBM Systems provides similar solutions[13].

Building ML platforms from open-source modules based on IaaS cloud and software containers with generic cloud orchestrators (e.g. Terraform, MiCADO/Occopus[14]) is another and feasible option that offers several benefits compared to above described MLaaS approaches, e.g. less vendor-locking, higher level of (security) control, etc. All these approaches leverage mostly on

open source tools and frameworks[15] [16], such as TensorFlow or Apache Spark (see HDInsight from MS Azure or Dataproc from GPC). Concentrating on the manufacturing sector as one of the most advancing application areas, a reference architecture has been recently proposed by Fraunhofer IOSB[17]. It was designed for scalable data analytics in smart manufacturing systems and complies with the higher-level *Reference Architecture Model for Industrie 4.0 (RAMI 4.0)*. The new reference architecture was implemented and validated in the Lab Big Data at the SmartFactoryOWL based on various open-source technologies (Spark, Kafka, Grafana). As machine learning tools and methods are embedded as the analytical components in many Big Data systems, the Big Data related reference architectures were also considered as a baseline in our project[18].

According to our findings, reasoning on Big Data (with ML), especially on the execution, facilitation and optimisation of such frameworks/applications on clouds can be significantly enhanced. Better support for the creation/assembly of ML applications from pre-engineered proprietary (MLaaS) or open-source building blocks can drive to a new generation of reference architectures that can be deployed on an orchestrated platform, leveraging on the available software components from repositories. Unlike these approaches, the presented novel reference architectures will be provisioned and orchestrated on the targeted infrastructure elements in a highly automated way, i.e. our work is to fill the gap between the theoretical frameworks and their actual deployments in a more effective way.

3 | COMPONENTS AND SERVICES OF THE BIG DATA AND ML FRAMEWORK

3.1 | MTA cloud and Occopus

MTA Cloud was founded by the Wigner Data Center and the Institute for Computer Science and Control (MTA SZTAKI) in 2015 as 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 be easily built based on predeveloped and published Occopus infrastructure descriptors[19]. 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)[3]. The novelty of Occopus was described and compared with similar cloud orchestrators[20]. Here we mention only one of its main advantages. The plugin architecture enables the use of modules for various cloud systems and hence AI reference architectures created by Occopus are easily portable among various cloud systems like Amazon AWS, Microsoft Azure, OpenStack, OpenNebula and CloudSigma.

3.2 | Support for parallel data storage and processing – Apache Hadoop

Apache Hadoop is an open source framework that enables the fast and easy deployment of data-intensive applications. It facilitates the development of commodity hardware based clusters. It uses a special data storage method, which is based on a distributed file system (HDFS, Hadoop Distributed File System[21]), that makes it possible to process efficiently terabytes of data in just minutes.

HDFS uses the MapReduce[22] 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. In general a file is split into several blocks of data which are stored separately on DataNodes. It is the NameNode that controls and coordinates the data storage by providing a map about the datablocks in the cluster while the Jobtracker's task is to coordinate the parallel processing of the data. The slave nodes make up the bulk of the machines, each of them running a DataNode and a TaskTracker daemon in order to be able to communicate with the Master node.

The Occopus infrastructure descriptors for such a Hadoop/HDFS cluster have been developed in MTA SZTAKI and are published on the web page of MTA Cloud[3] as well as on the web page of Occopus[6].

3.3 | Support for high performance, distributed data processing – Apache Spark

Apache spark[23] is an open source, distributed cluster-based system that was developed with the ultimate goal of accelerating and facilitating computations on Big Data. As opposed to Hadoop's Map reduce paradigm[22] it mainly stores data in memory, which enables up to ten times faster data processing. It was written in Scala and have different easy-to-use APIs, like Scala, Java, python and R. From an engineering perspective these APIs provide the biggest advantages and reason why choosing the Spark framework. It also has some built-in libraries, that facilitate data analysis and the usage of machine learning methods MLlib[24]. Spark was developed with the desire to enhance flexible configurations, to be run as standalone or in cluster mode, and can access data stored on different storage systems such as HDFS, Apache HBase, or Amazon S3, etc.

3.4 | Spark Machine learning library

Apache Spark MLlib[24] is the Apache Spark machine learning library that consists of commonly used learning algorithms and tools. As the core component library in Apache Spark, MLlib offers numerous supervised and unsupervised learning algorithms, from logistic regression to k-means 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[3] and on the web page of Occopus[6].

3.5 | 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[25] 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 does not require to be connected 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; code, data and file sharing; sharing computational resources available on a server among multiple users; centralized controlling of data access; centralized installation and configuration of R, R packages and other libraries.

Jupyter Notebooks[26] are starting to become extremely popular especially in education and field of empirical research. 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. Jupyter's popularity stems from supporting the so-called "literate programming" concept, which means, that codes and mathematical equations can be enriched with explanations or even multimedia resources. Moreover, structuring, editing, and displaying mathematical equations are simplified with MathJax support. Code sharing, result reproducibility, and version control is also a quite easy and widespread used practice in JSON format with GitHub. The built-in techniques like code completion or fast and easy help access promote Jupyter notebooks in the scientists' society.

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:

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

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

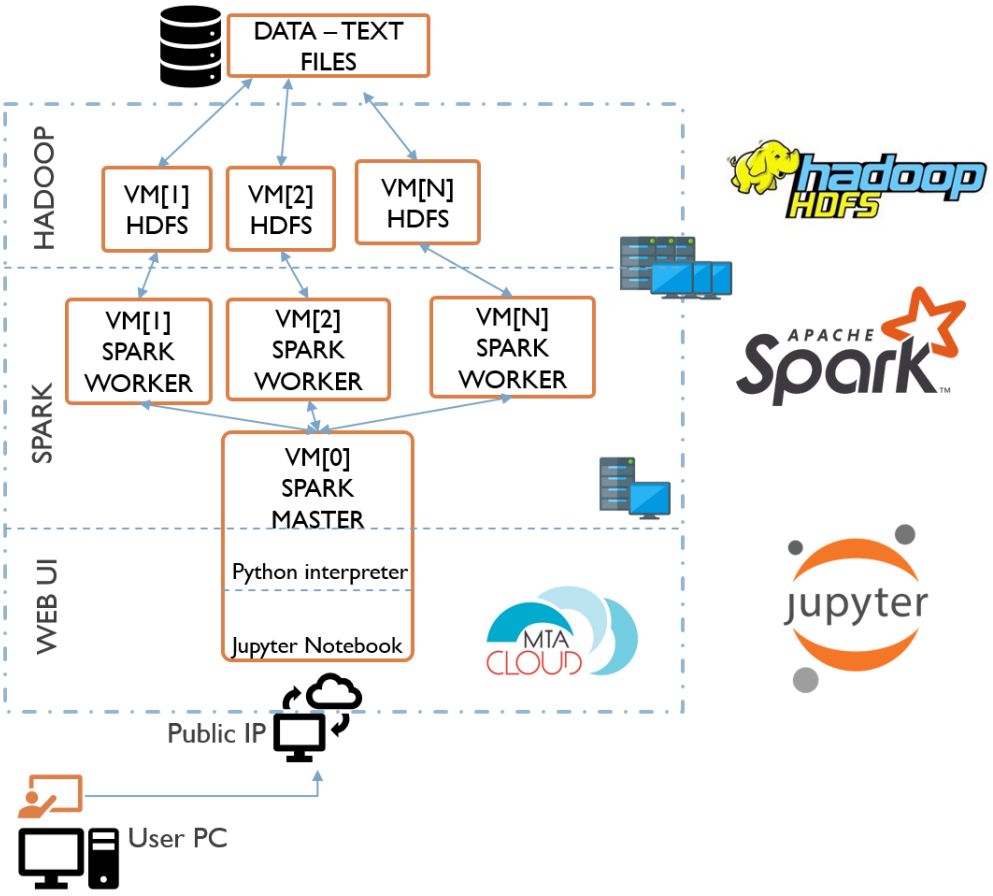

FIGURE 1 Apache Spark-based reference architecture for machine learning using Jupyter as the User Interface

4 | VALIDATION WITH TEXT CLASSIFICATION APPLICATION

The third step in the presented work is to apply the designed reference architectures for various big data and machine learning application domains. In this paper, we have selected the text classification domain to illustrate the application of the Spark-oriented reference architecture.

In many real-world scenarios, it is especially important and highly desired to automatically classify documents into a fixed set of categories. Common scenarios include classifying a large amount of archival documents such as newspaper articles, and research papers. Natural Language Processing offers a wide variety of methods for automatically classifying documents.

MTA Centre for Social Sciences seeked a solution for the following problem, using the computational power provided by MTA Cloud. The coding of public policy major topics on various legal and media corpora can be an important input for verifying different hypotheses and models in political science. This fundamental work has till recently mostly been conducted by double-blind human coding, which is still considered to be 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 codings, such as dictionary-based and supervised learning methods[27], offer a solution to these problems[28]. 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 this task described in Section 3.5, and at the same time demonstrated how to use this architecture for solving the problem. After the demonstration, the researchers of the Institute for Political Science started to use the RStudio version of the framework, meanwhile we have also investigated possible solutions for the problem using the Jupyter Notebook version[29]. In this section our approach is presented in details. The steps of solving the above described text classification problem are shown in Figure 2. 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

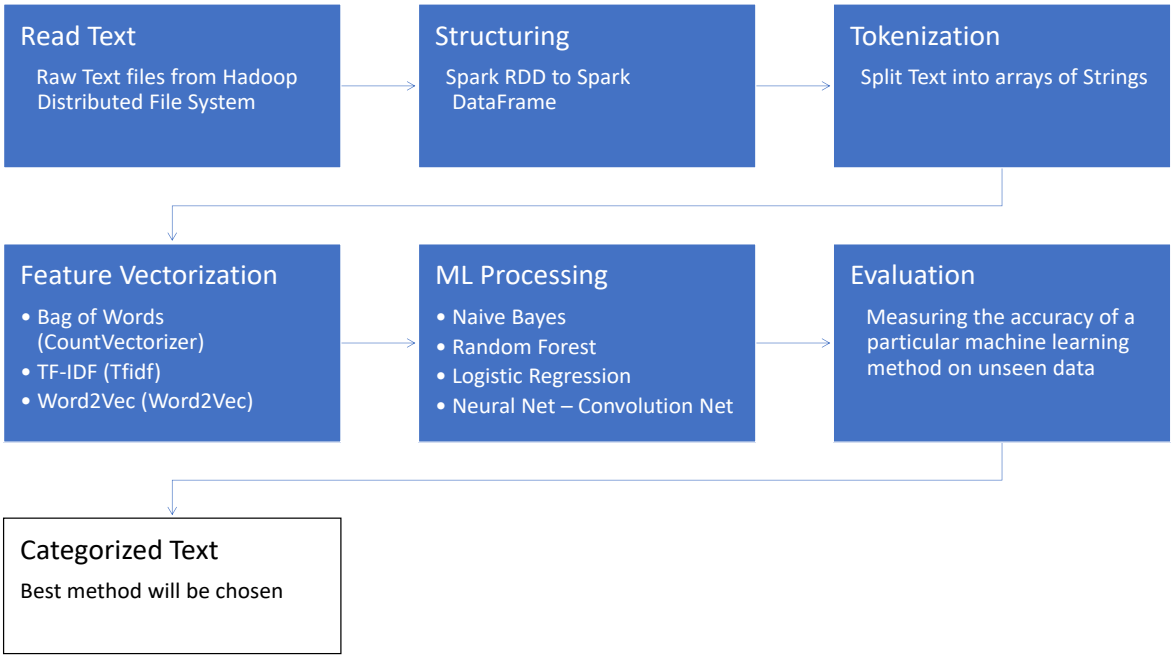

FIGURE 2 The proposed text processing pipelines based on HDFS and Apache Spark. It is worth mentioning, that the independent pipelines could be executed simultaneously.

and evaluate the different classification pipelines in parallel. In the following sections the different stages of our Spark-based pipelines are detailed.

4.1 | Data distribution

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 result of logically partitioning the original dataset and is the basic data structure of Spark. It is called resilient, or with other words fault tolerant, since Spark automatically takes care of handling the emerging failures. These logical partitions may be processed in parallel on different nodes of the cluster. As it was already described in Section 3.3, Apache Spark uses a parallel data processing method with storing the data mainly in memory. This results in a very fast data processing even 10 times faster than Hadoop Map Reduce's processing. Moreover, the uploading process is a very easily manageable task since it is done automatically by the Apache Spark - HDFS system without user intervention. After this distribution step, the uploaded data can be seen as a hierarchical set of files on HDFS.

4.2 | 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 or Python. A DataFrame contains rows with Schema. It can scale from kilobytes of data handled on a single laptop to petabytes of data processed on a large cluster. A DataFrame can be operated with the use of relational transformations such as filter, select, group by, sort, etc. Like Apache Spark in general, Spark SQL, in particular, enables the scalability of distributed in-memory computations.

4.3 | Data description

The original data set consisted of different articles from the two most popular Hungarian daily newspapers Népszabadság and Magyar Nemzet covering the period between 2010 and 2014[30]. The total corpora consisted of 37,143 articles which had been categorized by double-blind human coding and 31 different categories were established. We partitioned the available data into a training set having 70% of the available data, and to a test set containing the remaining 30%.

After conducting a brief statistical investigation on the whole corpora we got:

- the average number of words per document is 225.3;
- the standard deviation of the number of words is 174.5;
- the number of words in the shortest document is 22;
- the number of words in the largest document is 892;
- the distribution of the number of words is not normal.

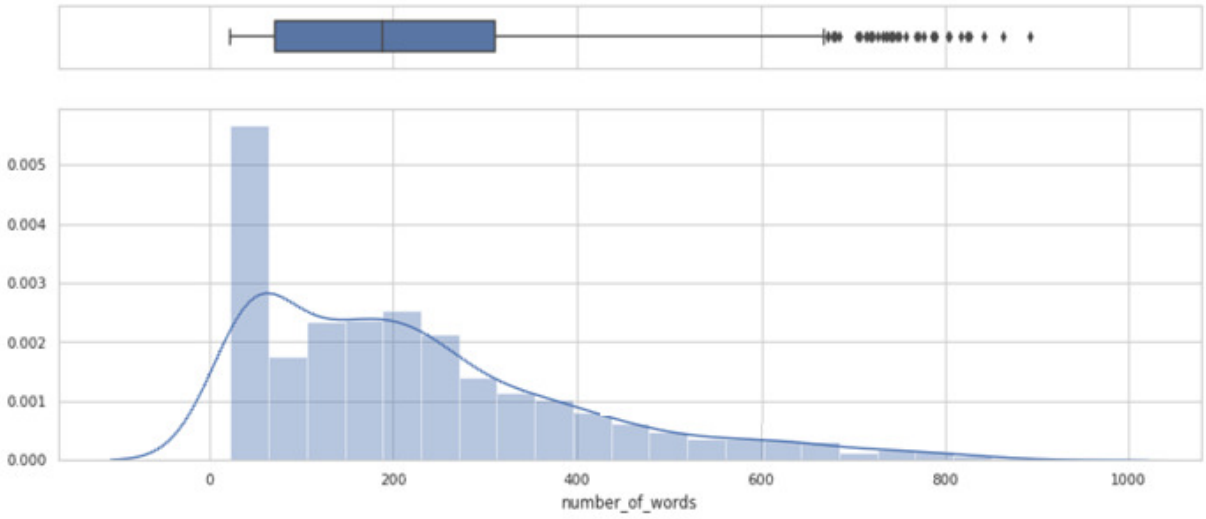

FIGURE 3 Histogram and kernel density estimate (KDE) of the number of words in the documents

In Figure 3 the distribution of the number of words in the documents can be seen together with the kernel estimate that gives information about the shape of this distribution.

Focusing only on the three richest categories (that involve the most documents): there are 7,668 documents in the largest category, 5,040 in the second largest and the third category involves 3,437 documents. Also, let us consider all the words that appear in the 37,143 documents. Figure 3 shows the distribution of the number of words in the documents. The x axis shows the number of words, and the histograms connected to a number shows the occurrences of such long documents in one category.

The number of words in the documents belonging to the richest category (containing the most documents) the average number of words is 140.9 (std dev. 154, min. 22 max. 892), so it can be concluded that these documents are mainly short ones. In the second category also short and long documents are classified, here the average number of words is 212.3 (std dev. 170, min. 22, max. 888) while in the third category the average number of words is 270.2 (std dev. 170, min. 22, max. 852). Based only on

the number of words that are present in the document the category cannot be determined. Therefore different Machine Learning algorithms were used for classification.

4.4 | 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 removal, 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 pieces, called tokens with a given space delimiter in the same way as a natural language speaker would do. Spark machine learning library also supports text mining with a wide variety of built-in functions, like RegexTokenizer. Therefore, users of the Spark environment shown in Figure 2 do not have to develop any new software for tokenization, just use the Spark ML RegexTokenizer function.

It is important to understand that while tokenization is a widely used practice in text classification scenarios, it is not as straightforward in case of the Hungarian, due to the structure of the language which is fundamentally different from Indo-European ones. The Hungarian being an agglutinative language the suffixes of a noun can be glued after each other without any separator. Nowadays we know about 40,000 different Hungarian words, from which we use about 2,000-4,000 in everyday communication. Due to the fact that tokenization discovers each noun as a separate word which stems are the same but the endings are different as a result the tokenization generated 3,720,000 different independent words.

TABLE 1 Illustration of tokenization issues with the hungarian language: these prepositions illustrated in english are suffixes in hungarian.

English version	***Hungarian version***
the *dog*	a *kutya*
with the *dog*	a *kutyá*val
to the *dog*	a *kutyá*nak
towards the *dog*	a *kutyá*hoz
from the *dog*	a *kutyá*tól

To highlight the problem in more detail in Table 1 a short but very frequent example can be seen. With this example, it can be understood that after tokenization in Hungarian so many words are generated, while in English only one: 'dog'. After receiving the available corpora from the Institute for Political Science at Centre for Social Sciences containing 37,143 documents we found that tokenization would generate 3,720,000 independent words.

To deal with this issue, we decided to select the most frequently used words. For starters, we defined an upper limit $\boldsymbol{N}$, and according to this limit, we only considered the $\boldsymbol{N}$ most frequently appearing word in the corpus having 37,143 documents. This number could be controlled by the Apache Spark ML directory, but we could also find some good settings that presented better results compared to other solutions. Choosing $\boldsymbol{N}$ too small would result in information loss concerning those words that have key roles in the classification but its occurrence is too rarely while having a larger $\boldsymbol{N}$ we encountered higher computation capacity requirement and did not manage to reach better accuracy, in this case, higher true classification rates.

Alternatively, hungarian specific linguistic processing tools could be applied. At first the stems of each word should be found and then only the stems should be considered. Researchers from Szeged, Hungary have created a promising tool named *magyarlanc*[31]. The toolkit is based on Java, therefore the implementation in the current ecosystem is difficult. Other solutions for stemming and lemmatization are based on regular expressions or other well-defined algorithms. The so-called Lancaster algorithm[32] could be extended with rules to allow stemming in hungarian. We experimented with Python NLTK (Natural Language Toolkit), but unfortunately we did not manage to derive expected accuracy.

Based on all these considerations we went through all the steps of text classification, and finally we managed to find a method that led to better performance.

4.5 | 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, these features should be represented in the form of vectors. Accordingly, the next stage in our pipeline of Figure 2 is feature vectorization. There are different kinds of feature vectorization algorithms and many of them are supported by the SparkML library. In the next sections, the applied feature vectorization and word embedding methods are briefly introduced.

4.5.1 | Bag-of-Words

The Bag-of-Words (BoW) model provides feature extraction capabilities. As the name suggests, it does not keep the words structured it just keeps a "bag" of the words. It gives back a histogram of the words within the text, i.e., considering the number of occurences of each word 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 text documents to vectors that represents on one hand the vocabulary of that documents and also the occurrences of each word in a specific document.

The biggest challenge with the Bag-of-Words model lies in the fact that based on our corpus it would produce a huge matrix having 3,720,000 columns and 37,000 rows as a result. The rows represent the documents related to the corpus and each column represents one word. A value of 1 in the matrix indicates that the word exists in that document, while having a 0 means that word is missing from the actual document. This matrix serves as an input of the learning algorithm. Since this method does not differentiate words that look very similar (e.g.: politikus, politikusnak, politikustól, …) thus the learning algorithm will also find complex and hard to generalize rules.

4.5.2 | TD-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a feature vectorization method widely used in text mining[33]. It not only reflects the frequency of occurrences of each word but gives information also about 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 to highlight documents that are closely related to a term but not to an exact string-match. The Spark ML function that supports this method is called IDF.

Although the TF-IDF algorithms assign weight values to the words based on the frequency and the importance of them, the TD-IDF generated matrix still would be the same size as in the case of the Bag-of-Words model.

4.5.3 | Word2Vec

The above mentioned two feature vectorization methods have also another shortcoming, namely the BoW and TF-IDF methods does not hold any information about the semantics of the word.

Word embedding techniques try to convert large one-hot vectors into vector representations with lower dimension, while preserving the meaning or its lingual context. The lingual context means the $\boldsymbol{m}$ surrounding words in this present scenario.

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 Word2Vec method was developed and published in 2013[34]. It is practically a method that uses a shallow neural network with 2 layers to generate unique vectors from input words with a predefined length. Common values for the representation dimensionality are between 150 and 500, in our experiments we chose to embed words as 300-dimensional vectors.

The context of a word has a key role in Word2Vec, because words that have similar contexts in the documents tend to have similar meanings and therefore their vector representations will be similar[33]. The aritmetic properties of such word embeddings are also well-defined[35].

The built-in Word2Vec algorithm uses the skip-gram neural network model. In the skip-gram model version of Word2Vec, the concept is to pick one word from the document and then to predict the words that are in the lingual context of the current word. This is a learning process, that is performed by a neural network with one hidden layer consisting of 300 neurons.

After an appropriate learning phase the end product of this learning will be an embedding layer in a network. It can be regarded as a kind of lookup table, where the rows are vector representations of each word in the documents' vocabulary. Given enough data, usage, and contexts, Word2Vec can predict a word's meaning based on past appearances with high accuracy. From now

on, these 300 unit-long vectors will represent a word throughout our experimentation which will serve as the input for machine learning algorithms.

In order to investigate the effectiveness of it we compared a few words for which the cosine of the vectors' angle is close to 1. Cosine similarity[36] is a widely used method for comparing vectors. In contrast to the Euclidean distance, cosine distance (which is the basis of cosine similarity) is not affected by vector magnitude, which could be affected by word frequency. Cosine similarity is based on the angle of the vectors, more independent to difference in occurrence count.

Table 2 shows a few examples we could find after the training phase. In this example we looked for the closest words in meanings to 'orvos' (doctor) based on vector distance. The results are shown in descending order in Table 2.

TABLE 2 Most similar words to doctor (*orvos* in hungarian). Words and cosine similarity of the representing vectors are given in descending order

Hungarian word	***English meaning***	***Cosine similarity***
orvosok	doctors	0.891
nővér	nurse	0.860
műtős	surgery	0.845
kórház	hospital	0.801
ambulancia	ambulance	0.789
egészség	health	0.753

We have conducted similar experiments with not just nouns but also with famous person's names. Though the results shown in Table 3 are subjective, we have found them satisfying.

TABLE 3 Names and titles related to *Orbán* (the surname of the hungarian Prime Minister). Words and cosine similarity of the representing vectors are given in descending order

Hungarian Surname	***Title***	***Cosine similarity***
Orbán	Prime Minister	1.000
Gyurcsány	Ex. Prime Minister	0.791
Rogán	Leader of Cabinet	0.731
Kövér	Chaiman of Parliament	0.729
Matolcsy	Chairman of Nat. Bank	0.711
Draskovics	Ex. Financial Minister	0.704

We would like to add, that similarly to words, documents could be represented as multi-dimensional vectors. A method defined by Le and Mikolov[37] results in a document representation, independent from the document length and word count. This technique called Paragraph Vector is implemented as Doc2Vec[38], available in popular libraries. We intend to extend the presented pipeline with this method in the future.

The application of the Word2Vec method raises another problem. Our main task was to determine which predefined class a document belongs to based on the existence or absence of words. It is important to note that the documents differ in length. We have not focused on the number of words a document involves yet, but with the help of some basic statistical indicators we will show that not only the different categories differ in the number of words but also those assigned to the same category as well.

4.6 | 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.

4.6.1 | Random Forest

Random forests are ensembles of decision trees[39]. Decision trees and their ensembles are very popular methods for classification and regression type tasks, since they are easy to interpret, can handle categorical features, can be extended to the multiclass classification setting, and are able to capture non-linearities and feature interactions[33]. The spark.ml implementation supports decision trees for binary and multiclass classification and for regression. The implementation partitions data by rows, allowing highly distributed training. In spark.ml Decision Tree classifier is available via the DecisionTreeClassifier() method[40].

To avoid overfitting, Random forests combine many decision trees. It trains a set of decision trees separately, so the training can be done in parallel. In order to involve randomness into the training process each decision tree is a bit different. The predictions from each tree are combined than which reduces the variance of the predictions and thus improves the performance on test data. In spark.ml implementation random forests are available via RandomForest() method[41]. There are various possibilities on how to set the parameters with this Apache Spark implementation of the Random Forest method. We have conducted several experiments changing the different parameters and we got the best accuracy on the test data with the following settings: 30 random generated trees, 20 level depth search with maximum 32 bins on one level, and 1 minimum instance per node.

Although Random Forest generates more than one tree and by aggregating them presents the ultimate result, each word can be measured to what extent they contributed to the classification. In other words which phrases played an important role in separating that category. It can be also very important for those scientists who are eager in understanding the grammar model in a deeper level.

4.6.2 | Naïve Bayes

Naïve Bayes classifiers are very widespread and successful tools to be used in text classification scenarios[42]. They use word count as a feature (in multinomial Naïve Bayes) or a zero/one value that indicates whether the word was found or not in the document respectively[33]. Naïve Bayes method in SparkML is based on Bayes's theorem for computing the conditional probability distribution of each feature given each label[40].

4.6.3 | Multinomial logistic regression

In terms of its structure, logistic regression can be thought of 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 was equal with the number of words coming from the bag of words, the tf-idf model[40]. Multiclass classification can be carried out with multinomial logistic (softmax) regression. In multinomial logistic regression, the algorithm generates K sets of coefficients, or a matrix of dimension $K \times J$, where K is the number of classes and J is the number of features. The conditional probabilities of the outcome classes are modeled using the softmax function.

The biggest advantage of logistical regression is that for each word we get a coefficient (beta) that tells us to what extent that word discriminates between the individual categories. In other words, this coefficient is a measurement of the importance from a classification point of view. The results generated by this logistical regression can be seen in Table 4.

TABLE 4 Results of logistical regression, ordering the words based on their importance in descending order

Word	English meaning	Word index	Importance
fájdalom	pain	75940	27.37
állatkertek	zoos	43920	23.87
robbanószer	explosion	71092	17.72
tőzsdék	stock markets	9992	13.05
AFP	AFP	6691	12.70
tárgyalás	negotiation	9304	12.35
kijelent	to declare	6603	12.26

4.6.4 | Artificial Neural Network

We have aggregated the word vectors of each word in a document, we have calculated the mean of them 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 a fully connected neural network (or feed-forward artificial neural network). A neural network 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.

4.6.5 | Convolutional Neural Network (CNN)

It is very important, that when we feed data to a CNN, each word vector should be 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 the same length of vector representation.

A neural network model will expect that all input data should have the same dimension. Having different documents containing different number of words (see section 4.3) results in different vector representations. One solution can be to define the maximum lengths of these vectors and padding with zeroes the vectors representing shorter documents, and truncating the data from richer documents. In other words, each document is represented as a fix-sized matrix, where rows are the words and the columns are the Word2Vec features. This transformation enables our data to be fed into a Convolutional Neural Net (CNN).

4.7 | Experiments and Evaluation

As a final step we tested, evaluated and compared the used classification models and then chose the best algorithm to classify the new incoming documents. All the experiments were conducted on an 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.

In our experiments, we were able to combine all the feature vectorization methods with all machine learning algorithms (shown in Fig. 1) with 2 exceptions:

1. In Naïve Bayes method feature values must be non-negative while the Word2Vec method produces real numbers.

TABLE 5 Results of the different feature vectorization - machine learning experiments

Method's name	Type of Tokenization	Evaluation time (sec)	Accuracy on Training data	Accuracy on Test data
Logistic Regression	BoW	15.39	99.4%	65.5%
Naïve Bayes	BoW	0.74	80.7%	47.6%
Random Forest	BoW	148.17	60.7%	47.5%
ANN(*)	BOW	210.32	74.2%	65.7%
ANN(*)	TF-IDF	238.27	99.9%	58.4%
ANN(**)	BoW	431.65	87.5%	64.6%
ANN(**)	TF-IDF	431.65	99.9%	58.6%
ANN(**)	W2V	251.15	80.3%	58.9%
Naïve Bayes	TF-IDF	0.82	78.7%	42.1%
Random Forest	TF-IDF	168.29	47.7%	29.9%
Random Forest	W2V	29.49	85.3%	60.8%
ANN	W2V	132.17	90.0%	59.4%
CNN	W2V	110.0	97.0%	**78.8%**

2. Convolution neural network 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 the convolutional network, the bag-of-word, and TF-IDF methods can not.

We have conducted experiments using two different kinds of artificial neural networks. ANN(*) Spark ML Neural Network was used with 1 hidden layer consisting of 200 neurons, while ANN(**) had 2 hidden layers with 220 and 200 neurons respectively.

The CNN Convolutional Network was used with the following settings: on the first layer with 100 filters having a dimension of 1x10, on the second layer with 160 filters with the same 1x10 dimension, the third dropout layer with a probability of 0.5 (or 50 % rate) and finally a fully connected layer with 31 output neurons (because we had 31 categories). We used Adam optimizer.

We based our evaluation on the measured accuracy for the test data, or in other words how close the outputs of the machine learning algorithms are to the true values, in this case, how many percents of the categorizations were correct. For the sake of curiosity, we also presented the measured values for all feature vectorization-machine learning combinations with the execution time as well.

Based on the results shown in Table 5 we found that while the shallow neural network with TF-IDF gave the highest accuracy on the training set, the Word2Vec feature combined with the Convolutional Neural Network algorithm gave the best performance with an accuracy of 78.8% on the test data.

It is important to point out, that the measured accuracy on the training set does not represent the classificational ability of the model itself, it is only included to show how well the model is able to fit the data. Therefore, we can conclude that the method of Random Forest and TF-IDF results in underfitting.

The high performance of convolutional networks for text classification is well-known[43], it has been proven that features encoded in word vectors could be extracted using convolutional filters to support classification. It is understandable, that the fully-connected layers of the ANN have less representational ability compared to the convolutional filters of the CNN, which explains the outcome.

Is is also interesting to analyze that by terms of test classification performance, the ANN with BoW method outperformed the ANN with W2V model. The cause behind this phenomena is based on the nature of fully-connected artificial neural nets. The sparse representation of the BoW tokens are somewhat ideal feature extraction, while the dense and compressed word vectors need more feature extraction ability. We would like to note, that a deep neural network with a larger number of hidden layers and elements could perform better, i.e. the semantic properties of words could be taken into account on classification.

It is also worth mentioning, that the BoW representation is memory heavy, while the W2V embedded representation is itself applying a dimensionality reduction[44]; therefore, the tradeoff between computational complexity (memory cost and runtime) and model performance (classification accuracy) is observable on these two cases.

5 | CONCLUSION AND FUTURE WORK

Leveraging on the concept of reference architectures, we have created a big data and AI application framework that needs three major steps to be performed:

1. Occopus orchestrator tool to define and deploy the required infrastructure in the target cloud. This tool was released by MTA SZTAKI under the Apache Licence.

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 MTA SZTAKI[45], and can be used according to the actual AI application class.

3. An actual deployed instance of the application-oriented big data and AI application development and execution framework that is built by Occopus (Step 1) according to the Occopus infrastructure descriptors (Step 2) 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 and tailor a selected reference architecture for big data and machine learning application development and execution with a special focus on 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 Figure 2.

The Apache Spark architecture proved to be a good choice because with the off-line or batch learning method trained machine-learning algorithms can also be used in real-time with the Apache Spark Streaming API without training the model again. Further advantage of our solution is that it can be scaled arbitrary according to the increased load without any change in the code itself. Therefore, the online news could be categorized in real time.

Although the presented big data and machine learning application development and execution framework was created and tested on MTA Cloud it can be used on other clouds including Amazon AWS, Microsoft Azure, OpenStack, OpenNebula and CloudSigma due to the plugin architecture of the underlying Occopus cloud orchestrator[20]. 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 (see Section 3.5).

Future work will intend to create further AI reference architectures to cover other AI application classes. On the other hand, the training phase was executed on a moderate size of data set (37,143 articles from the two most popular daily newspapers of Hungary between), the parallel and distributed environment is especially important for further applications since, in the future we would like to extend our investigations also on online media corpora and online news, as they are gaining ground all over the world.

6 | ACKNOWLEDGEMENT

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. The presented work was partially funded by the European H2020 NEANIAS project under grant No. 863448, by the Hungarian Scientific Research Fund (OTKA) under project No. 132838, and by Bolyai+ Scholarship for Young Higher Education Teachers and Researchers under grant No. ÚNKP-20-5-OE-73.

References

1. Nagy E, Hajnal Á, Pintye I, Kacsuk P. Automatic, cloud-independent, scalable Spark cluster deployment in cloud. *CIVIL-COMP PROCEEDINGS* 2019; 112: 1–9.

2. Lovas R, Koplányi K, Élö G. Agrodat: A Knowledge Centre and Decision Support System for Precision Farming Based on IoT and Big Data Technologies. *ERCIM News* 2018; 113: 22–23.

3. MTA Cloud. https://cloud.mta.hu/; Accessed: 2020-05-14.

4. Castillo F.-dE, Scardaci D, García ÁL. The EGI federated cloud e-infrastructure. *Procedia Computer Science* 2015; 68: 196–205.

5. Whitepapers – Amazon Web Services (AWS). https://aws.amazon.com/whitepapers/; Accessed: 2020-05-14.

6. Occopus. http://occopus.lpds.sztaki.hu; Accessed: 2020-05-14.

7. Semberecki P, Maciejewski H. Distributed classification of text documents on Apache Spark platform. In: Springer. ; 2016: 621–630.

8. Pranckevičius T, Marcinkevičius V. Application of logistic regression with part-of-the-speech tagging for multi-class text classification. In: IEEE. ; 2016: 1–5.

9. Science Cloud. https://science-cloud.hu/felhasznalastsegito-; Accessed: 2020-11-10.

10. Pintye I, Kail E, Kacsuk P. Big data and machine learning framework for clouds and its usage for text classification. In: 11th International Workshop on Science Gateways (IWSG 2019), 12-14 June, 2019, Ljubljana, Slovenia (in press). .

11. Azure Reference Architectures. https://docs.microsoft.com/en-us/azure/architecture/reference-architectures; Accessed: 2020-05-14.

12. HPE Reference Architecture for AI on HPE Elastic Platform for Analytics (EPA) with TensorFlow and Spark, White Paper, HPE, 2018. https://assets.ext.hpe.com/is/content/hpedam/documents/a00060000-0999/a00060456/a00060456enw.pdf; Accessed: 2020-05-14.

13. Lui K., Karmiol J. AI Infrastructure Reference Architecture IBM Systems, 87016787USEN-00, 2018. https://www.ibm.com/downloads/cas/W1JQBNJV; Accessed: 2020-05-14.

14. Kiss T, Kacsuk P, Kovacs J, et al. MiCADO - Microservice-based Cloud Application-level Dynamic Orchestrator. *Future Generation Computer Systems* 2019; 94: 937 - 946. doi: https://doi.org/10.1016/j.future.2017.09.050

15. Nguyen G, Dlugolinsky S, Bobák M, et al. Machine Learning and Deep Learning Frameworks and Libraries for Large-Scale Data Mining: A Survey. *Artif. Intell. Rev.* 2019; 52(1): 77–124. doi: 10.1007/s10462-018-09679-z

16. Pop D, Iuhasz G, Petcu D. *Distributed Platforms and Cloud Services: Enabling Machine Learning for Big Data*: 139-159; 2016

17. Al-Gumaei K, Müller A, Weskamp JN, Longo CS, Pethig F, Windmann S. Scalable Analytics Platform for Machine Learning in Smart Production Systems. In: 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). ; 2019: 1155-1162.

18. Paakkonen P, Pakkala D. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research* 2015; 2(4): 166 - 186. doi: https://doi.org/10.1016/j.bdr.2015.01.001

19. Lovas R, Nagy E, Kovács J. Cloud agnostic Big Data platform focusing on scalability and cost-efficiency. *Advances in Engineering Software* 2018; 125: 167–177.

20. Kovács J, Kacsuk P. Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. *Journal of Grid Computing* 2018; 16(1): 19–37.

21. Borthakur D, others . HDFS architecture guide. *Hadoop Apache Project* 2008; 53(1-13): 2.

22. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. *Communications of the ACM* 2008; 51(1): 107–113.

23. Zaharia M, Xin RS, Wendell P, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM* 2016; 59(11): 56–65.

24. Meng X, Bradley J, Yavuz B, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 2016; 17(1): 1235–1241.

25. Open source and enterprise-ready professional software for data science - RStudio. https://www.rstudio.com/; Accessed: 2020-05-14.

26. Perkel JM. Why Jupyter is data scientists' computational notebook of choice. *Nature* 2018; 563(7732): 145–147.

27. Burscher B, Vliegenthart R, De Vreese CH. Using supervised machine learning to code policy issues: Can classifiers generalize across contexts?. *The ANNALS of the American Academy of Political and Social Science* 2015; 659(1): 122–131.

28. Albaugh Q, Soroka S, Joly J, Loewen P, Sevenans J, Walgrave S. Comparing and combining machine learning and dictionary-based approaches to topic coding. In: 7th annual Comparative Agendas Project (CAP) conference. ; 2014: 12–14.

29. Sebők M, Kacsuk Z. Classifying newspaper articles with the hybrid binary snowball process. In: POLTEXT 2019 Conference. ; 2019; Institute for Advanced Study, Waseda University, Tokyo.

30. Hungarian Comparative Agendas Project (CAP)- Media. https://openarchive.tk.mta.hu/399/; Accessed: 2020-05-14.

31. Zsibrita J, Vincze V, Farkas R. magyarlanc: A tool for morphological and dependency parsing of hungarian. In: RANLP 2013. ; 2013: 763–771.

32. Paice CD. Another stemmer. In: . 24. ACM New York, NY, USA. ; 1990: 56–61.

33. Apache SparkTM - Unified Analytics Engine for Big Data. https://spark.apache.org; Accessed: 2020-05-14.

34. Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* 2013.

35. Gittens A, Achlioptas D, Mahoney MW. Skip-Gram- Zipf+ Uniform= Vector Additivity. In: The 55th Annual Meeting of the Association for Computational Linguistics. ; 2017: 69–76.

36. Singhal A, others . Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.* 2001; 24(4): 35–43.

37. Le Q, Mikolov T. Distributed Representations of Sentences and Documents. In: ICML'14. ICML. JMLR.org; 2014: II–1188–II–1196.

38. Lau JH, Baldwin T. An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. In: Workshop on Representation Learning for NLP. ; 2016: 78–86.

39. Breiman L. Random forests. *Machine learning* 2001; 45(1): 5–32.

40. Classification and regression - MLlib main guide. https://spark.apache.org/docs/latest/ml-classification-regression.html.; Accessed: 2020-05-14.

41. Ensembles - RDD-based API - Spark 2.4.0 Documentation. https://spark.apache.org/docs/latest/mllib-ensembles.html; Accessed: 2020-05-14.

42. Kim SB, Han KS, Rim HC, Myaeng SH. Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering* 2006; 18(11): 1457–1466.

43. Kim Y. Convolutional Neural Networks for Sentence Classification. In: EMNLP. Association for Computational Linguistics; 2014; Doha, Qatar: 1746–1751

44. Kertész G. Metric Embedding Learning on Multi-Directional Projections. *Algorithms* 2020; 13(6): 133.

45. Laboratory of Parallel and Distributed Systems | MTA SZTAKI. https://www.sztaki.hu/en/science/departments/lpds; Accessed: 2020-05-14.