Harnessing the Internet of Everything (IoE) for Accelerated Innovation Opportunities

Pedro J.S. Cardoso  
*University of Algarve, Portugal*

Jânio Monteiro  
*University of Algarve, Portugal*

Jorge Semião  
*University of Algarve, Portugal*

João M.F. Rodrigues  
*University of Algarve, Portugal*
Advances in Computer and Electrical Engineering (ACEE) Book Series

Srikanta Patnaik
SOA University, India

ISSN: 2327-039X
EISSN: 2327-0403

MISSION
The fields of computer engineering and electrical engineering encompass a broad range of interdisciplinary topics allowing for expansive research developments across multiple fields. Research in these areas continues to develop and become increasingly important as computer and electrical systems have become an integral part of everyday life.

The Advances in Computer and Electrical Engineering (ACEE) Book Series aims to publish research on diverse topics pertaining to computer engineering and electrical engineering. ACEE encourages scholarly discourse on the latest applications, tools, and methodologies being implemented in the field for the design and development of computer and electrical systems.

COVERAGE
- VLSI Design
- Microprocessor Design
- Circuit Analysis
- Optical Electronics
- Chip Design
- VLSI Fabrication
- Power Electronics
- Applied Electromagnetics
- Computer science
- Electrical Power Conversion

IGI Global is currently accepting manuscripts for publication within this series. To submit a proposal for a volume in this series, please contact our Acquisition Editors at AcquisitionEditors@igi-global.com or visit: http://www.igi-global.com/publish/.

The Advances in Computer and Electrical Engineering (ACEE) Book Series (ISSN 2327-039X) is published by IGI Global, 701 E. Chocolate Avenue, Hershey, PA 17033-1240, USA, www.igi-global.com. This series is composed of titles available for purchase individually; each title is edited to be contextually exclusive from any other title within the series. For pricing and ordering information please visit http://www.igi-global.com/book-series/advances-computer-electrical-engineering/73675. Postmaster: Send all address changes to above address. Copyright © 2019 IGI Global. All rights, including translation in other languages reserved by the publisher. No part of this series may be reproduced or used in any form or by any means – graphics, electronic, or mechanical, including photocopying, recording, taping, or information and retrieval systems – without written permission from the publisher, except for non commercial, educational use, including classroom teaching purposes. The views expressed in this series are those of the authors, but not necessarily of IGI Global.
Titles in this Series

For a list of additional titles in this series, please visit: www.igi-global.com/book-series

**Advancing Consumer-Centric Fog Computing Architectures**
Kashif Munir (University of Hafr Al-Batin, Saudi Arabia)

**New Perspectives on Information Systems Modeling and Design**
António Miguel Rosado da Cruz (Polytechnic Institute of Viana do Castelo, Portugal) and Maria Estrela Ferreira da Cruz (Polytechnic Institute of Viana do Castelo, Portugal)
Engineering Science Reference • copyright 2019 • 332pp • H/C (ISBN: 9781522572718) • US $235.00 (our price)

**Advanced Methodologies and Technologies in Network Architecture, Mobile Computing, and Data Analytics**
Mehdi Khosrow-Pour, D.B.A. (Information Resources Management Association, USA)
Engineering Science Reference • copyright 2019 • 1857pp • H/C (ISBN: 9781522575986) • US $595.00 (our price)

**Emerging Innovations in Microwave and Antenna Engineering**
Jamal Zbitou (University of Hassan 1st, Morocco) and Ahmed Errkik (University of Hassan 1st, Morocco)
Engineering Science Reference • copyright 2019 • 437pp • H/C (ISBN: 9781522575399) • US $245.00 (our price)

**Advanced Methodologies and Technologies in Artificial Intelligence, Computer Simulation, and Human-Computer Interaction**
Mehdi Khosrow-Pour, D.B.A. (Information Resources Management Association, USA)
Engineering Science Reference • copyright 2019 • 1221pp • H/C (ISBN: 9781522573685) • US $545.00 (our price)

**Optimal Power Flow Using Evolutionary Algorithms**
Provas Kumar Roy (Kalyani Government Engineering College, India) and Susanta Dutta (Dr. B. C. Roy Engineering College, India)
Engineering Science Reference • copyright 2019 • 323pp • H/C (ISBN: 9781522569718) • US $195.00 (our price)

**Advanced Condition Monitoring and Fault Diagnosis of Electric Machines**
Muhammad Irfan (Najran University, Saudi Arabia)
Engineering Science Reference • copyright 2019 • 307pp • H/C (ISBN: 9781522569893) • US $225.00 (our price)

**The Rise of Fog in the Digital Era**
K.G. Srinivasa (Chaudhary Brahm Prakash Government Engineering College, India) Pankaj Lathar (Chaudhary Brahm Prakash Government Engineering College, India) and G.M. Siddesh (Ramaiah Institute of Technology, India)
Engineering Science Reference • copyright 2019 • 286pp • H/C (ISBN: 9781522560708) • US $215.00 (our price)
Editorial Advisory Board

João Barroso, University of Trás-os-Montes e Alto Douro, Portugal
Carlos Tavares Calafate, Technical University of Valencia, Spain
Patrikakis Charalampos, University of West Attica, Greece
Delia Garijo, University of Seville, Spain
Viacheslav Izosimov, KTH Royal Institute of Technology, Sweden
Maksim Jenihhin, Tallinn University of Technology, Estonia
Marek Kubalčík, Tomas Bata University, Czech Republic
Andrew Patton McCoy, Virginia Polytechnic Institute and State University, USA
Marco Ottavi, University of Rome Tor Vergata, Italy
George Papadopoulos, University of Cyprus, Cyprus
Cristina Portalés Ricart, Universitat de València, Spain
Marcelino Santos, INESC-ID Lisbon, Portugal
Mario Schölzel, University of Potsdam, Germany & IHP in Frankfurt, Germany
João Paulo Teixeira, INESC-ID Lisbon, Portugal
Chapter 2
Application of Machine Learning Algorithms to the IoE: A Survey

Pedro J. S. Cardoso
https://orcid.org/0000-0003-4803-796
University of Algarve, Portugal

Jânio Monteiro
University of Algarve, Portugal

Nelson Pinto
https://orcid.org/0000-0002-8041-9199
University of Algarve, Portugal

Dario Cruz
https://orcid.org/0000-0001-9465-0845
University of Algarve, Portugal

João M. F. Rodrigues
University of Algarve, Portugal

ABSTRACT
The internet of everything is a network that connects people, data, process, and things, making it easier to understand that many subfields of knowledge are discussable while addressing this subject. This chapter makes a survey on the application of machine learning algorithms to the internet of everything. This survey is particularly focused in computational frameworks for the development of intelligent systems and applications of machine learning algorithms as possible engines of wealth creation. A final example shows how to develop a simple end-to-end system.

DOI: 10.4018/978-1-5225-7332-6.ch002
Application of Machine Learning Algorithms to the IoE

INTRODUCTION

The first steps toward the present Internet were made in the late 1950s, with the initial studies on packet switching. After that, the development of protocols for internetworking, by which multiple separate networks could be joined into a network of networks, were made. Later, in 1969, the first internetwork message was sent over the Advanced Research Projects Agency Network (ARPANET), from the University of California to a second network node at Stanford Research Institute. A definition came for the Internet as the worldwide interconnection of individual networks operated by government and other third parties. However, the very first commercial Internet Service Providers for the Internet we know and use today only appeared in the late 1980s, established in Australia and the United States. In the same decade, the earlier World Wide Web was devised with the linking documents conception, forming an information system reachable by any network node, consequence of the researches made at European Organization for Nuclear Research (CERN). Since then, the Internet usage has spread in such a way that the International Telecommunications Union estimates the number of world Internet users at 3.6 billion by end 2017, i.e., 48% of the world’s population.

In the Internet context, several, many times overlapping concepts, appeared in the last decades (Lueh, 2015; Perera, 2017): Machine-to-Machine (M2M), Internet of Things (IoT), Internet of People – IoP, Web of Things (WoT), Internet of Everything (IoE) etc. (see Figure 1). As the name suggests, M2M indicates the communication between machines over some mean and protocol. In the present, those communications many times are done using the Internet Protocol (IP). On the other hand, the IoP is the internet that connects people, delivering information generated by persons. The WoT has a narrower scope as it solely focuses on software architecture. IoT is more intricate to bound since many times it moves from the sensors, tags and actuators to the end users, passing through the deployment of electronics and firmware, communications, (embed, edge or datacenter) computation, data storage etc. (see Figure 2). Conjugating the characterizations presented in various works (Serpanos & Wolf, 2018; Tan & Wang, 2010; Vermesan & Friess, 2011), a long definition arises as: the IoT is the dynamic global network infrastructure, with self-configuring capabilities based on standard and interoperable communication protocols, where a massive number of physical and virtual things have identities, physical attributes, and virtual personalities. These things use intelligent interfaces, often over the same Internet Protocol that connects the Internet, to connect and communicate, without human-to-machine input, through wired and wireless networks within social, environment, and user contexts, being seamlessly integrated in smart spaces and into the information network. In other words, IoT allows people and things (physical devices, vehicles, buildings and other items embedded with electronics, software, sensors, tags, actuators, tags etc.) to be connected anywhere, anytime, with anything and anyone, enabling the collection and exchange of data. Extending the IoT concept, IoE aims to include all sorts of connections that one can envision as, the IoE is the networked connection of people, data, process, and things. In other words, IoE extends IoT by including intelligent and robust communication between machines-to-people (M2P), machine-to-machine, people-to-machines and people-to-people (P2P), i.e., the more expansive IoE concept includes M2M communications, machine-to-people and technology assisted people-to-people interactions.

It is expected that by 2020 between 20 and 30 billion devices will be interconnected in the IoT/IoE space. This rise in the quantity of apparatuses will also be accompanied by a growing diversity of distinct IoT/IoE device types, capable of directly gathering information from multiple sources, including health monitoring, asset tracking, environmental monitoring, predictive maintenance and home automation,
Application of Machine Learning Algorithms to the IoE

computers, vehicles, smartphones, appliances, jewelry, toys, wearables, building automation systems, and much more, ranging from consumer devices to industrial assets.

As already stated, equipped with connectivity, electronics (including sensors, tags and actuators) and software (to capture, filter and exchange data about themselves and their environment), IoT/IoE devices are a source of endless data, which can be used for the improvement of the systems that they populate. The huge amount of data generated by sensors must therefore be analyzed using proper methodologies in the search for patterns, to make predictions, to classify the data, detect outliers, detect security problems etc. This is where Machine Learning (ML) will have an important role. In its pure form, ML is supported in mathematical and computer science techniques to build models from given data, which are then applied to new and unseen data, in order to predict new outcomes (Alpaydin, 2016; Domingos, 2015; Witten, Frank, Hall, & Pal, 2016).

ML algorithms ordinarily allow to save resources by automatically analyzing data, which produces an expectably better overview of the available information, to make decisions that are more reasoned. But, where is ML applied? Imagine you decide to go to the cinema. You go to Google, click the “Search by voice” link and are asked to “Speak now… Listening…”. To which you reply, “films near me”. Behind the scene, a ML algorithm starts to work, translating your speech to text (Schalkwyk et al., 2010). Another system receives your order and searches the information system for proper answers to your question: “But, where does he/she lives? What are his/her interests? What time is it? How far is he/she from the nearest cinema rooms?” The result is an advised list of films, supported in yours and others preferences and actions. You choose the first one from the list, after all, the algorithm has just guessed what you want to see, and you have one hour until the session begins. Just enough time. While you leave the house, you say “Ok Google, turn off the lights” – again your speech is transformed into a command, turning off all the lights in the house. When you arrive to your car, the navigation app is activated, and you wait for a few instants while algorithms, supported on your previous query, street maps, predicted and real traffic etc., decide the best directions to the cinema. Before arriving the cinema, you are pulled over by a policeman who asks for documents and proof of insurance, the one that a ML algorithm helped to decide the quote based in your profile (Roy & George, 2017). In the cinema parking, you use the assisted self-parking from your car, based on sensors’ data and visual computing algorithms (Wang, Song, Zhang, & Deng, 2014). Finally, at the cinema you pay with your credit card, from a company that uses ML methods to attract new customers, drive up turnover, provide personalized recommendations, and detect frauds (Matsatsinis, 2002).

Focused on the described context, this chapter makes a survey on the application of ML algorithm in the IoE environment, mainly from the storage and machine learning point of views. The survey will be particularly interested in frameworks for the development of intelligent systems and ML applications. We should notice that other important issues are out of the scope of this chapter, such as cloud computing, cybersecurity, advanced analytics, connectivity and communication technologies, augmented and virtual reality, blockchain etc. Also out of the chapter’s scope is edge computing (see Figure 2), which offers significant computational advantages such as real-time decision making through edge analytics, reduced data transfer cost through compression and cleansing, improved security and data continuity through local operations.
This chapter is structured as follows. The second section, “Data Storage Perspectives and Applications”, will be devoted to the analysis of the information storing problematic. The section will start by analyzing different perspectives from local to global databases, with distinct storage systems (particular emphasis is given to relational and document oriented databases). A survey on publications about data storage in the IoT/IoE context will finalize the section. Next, the “Machine Learning Algorithms and Applications” section will address the ML problematic from an IoT/IoE perspective. A brief survey on the type of problems solved by ML is presented. Also addressed are the available tools to implement ML systems, finalizing with some applications of ML systems in the IoE context. The final sections draw some solutions, recommendations, future research directions and a conclusion.
Application of Machine Learning Algorithms to the IoE

DATA STORAGE PERSPECTIVES AND APPLICATIONS

In general, sensors produce Boolean, single numerical data (integer or floating-point numbers), or more complex information, possibly containing several forms of data, that is grouped together. For example, temperature sensors produce integer/float values with different precisions, depending on the application and sensor’s precision; Ambient sensors (can) read light intensity, temperature, humidity, and motion in a room; Global Positioning System (GPS) sensors return values such as latitude, longitude, altitude, or speed; and IP camera sensors produce arrays of values. The frequency of readings, the types of queries to be made on the data, and the quantity of data are also important factors, as they must be taken into consideration while deciding to store the data locally or globally. Therefore, the implementation of the database, i.e., the set of related data and the way it is organized, is an issue in the IoE field that should be carefully considered as it can deeply influence the performance of the system.

Storing data using the file system of local devices, such as a SD card, hard drive etc., can be difficult depending on what the data embodies. Therefore, many times the use of proper databases systems is highly recommendable, being those usually divided in two main types: relational and non-relational (or NoSQL) databases (Perkins, Redmond, & Wilson, 2018). The difference between the types of databases emerge in how they are built, the type of information they store, and how they store it. The software that implements the interactions with end-users, other software, and the database is called Database Management Systems (DBMS). Among its functions, the DBMS allows the administration of the database and the implementation of the basic create, read, update, and delete (CRUD) operations over the data.

In the case of relational databases, as main elements, they have entities/relations, many times seen as tables, and relationships between those entities. Each table’s row contains a tuple of information, identified by the table’s primary key. The relationships are maintained using a system of linked information from different tables through the use of primary and foreign keys, being divided in three types of relationships: one-to-one (exactly one record in the first table corresponds to none or one record in the related table), one-to-many (a primary table contains one record that relates to none, one, or many records in a secondary table), and many-to-many (each record in both tables can relate to any number of records in the other table). Examples of a one-to-one relationship, not very common as this case is usually solved by merging the two tables, would be between a sensor and its unique identification if they were placed in different relations; an one-to-many relationship example would be the link between a sensor and its readings; and a many-to-many relationship occurs between a set of mobile sensors and their location within a set of possible ones.

Structured Querying Language (SQL) is the common choice to communicate with relational DBMS (RDBMS), being implemented by the vast majority of those RDBMS programs. Storage can also be classified as local, global or hybrid, depending on the location of the data. Commercial and non-commercial application, among other, include PostgreSQL (2018), MySQL (2018), Microsoft SQL Server (Assaf, West, Aelterman, & Curnutt, 2018; Microsoft-SQL Server, 2017), or SQLite (2018). SQLite distinguishes itself from the rest by being a server-less RDBMS, i.e., a database engine that runs within the same process, thread, and address space as the application, where there is no message passing or network activity.

Relational databases, in general satisfy, four properties, namely: Atomicity, Consistency, Isolation and Durability (ACID). Atomicity points that a transaction is a logical unit of work which must either perform all data modifications, or none; Consistency means that all data must be left in a consistent state at the end of the transaction; Isolation means that data modifications made by a transaction are
independent of other transactions; and Durability implies that when a transaction is completed, effects of the modifications are permanent in the system. The DBMS that implement these four properties are termed as ACID compliant.

In our context, different sensors produce different forms of data, making it hard to create a well-structured database schema for that heterogeneous/dynamically structured set of data. In this sense, to store that unstructured information, it might be useful to use schema-less databases, making most of the adjustments to the database transparent and automatic. Furthermore, not all ACID properties are necessarily significant in the IoE context. In this sense, NoSQL systems have gained popularity for reasons such as the flexibility they provide in organizing data, obtained by relaxing the more rigid schemas stipulated by the relational model. The term NoSQL is used as an umbrella term for all databases and data storage that do not follow the RDBMS principles, being also often related to large data sets accessed and manipulated on a Web scale (Harrison, 2015; Perkins et al., 2018). We should notice that this flexibility does not mean that a schema should not be thought. On the contrary, the schema allows applications to know what and where is the information.

NoSQL databases are usually divided in four categories, namely: key-value pair storage, where every single item in the database is stored as a key and its value (e.g., Riak-KV (2018)); Wide column store, where data is stored in a columnar fashion (e.g., Cassandra (2018)); document store, where data is stored as semi-structured documents, typically in JSON (JavaScript Object Notation) or XML format (e.g., MongoDB (2018)); and graph databases, that use graph structures for semantic queries with nodes, edges and properties to represent and store data (e.g., Neo4j (2018)).

Among those databases, one of the most well-known schema-less database is, probably, the MongoDB database (Chodorow, 2013; Perkins et al., 2018). MongoDB presents a high performance, high reliability, easy scalability (vertically and horizontally through replication and auto-sharding, respectively) and map-reduce support. A MongoDB database is structured as a set of collections, which store documents. These documents are BSON objects (Bassett, 2015), a binary JSON document format supporting dynamic schemas, i.e., documents in the same collection are not forced to have the same structure.

Table 1 summarizes a comparison between relational databases and MongoDB terminology. Also, Figure 3 compares relational and document storage databases, and the main DBMS’s features are summarized in Table 2 (see Appendix).

Recently, RDMS such as MySQL started to support a native JSON data type defined by the RFC 7159 that enables efficient access to data in JSON documents. Among other, MySQL automatically validates JSON documents, stored in JSON columns, producing an error upon the insertion of invalid documents, which would not happen when storing JSON-format strings in a common string column. Furthermore, the storage is optimized by converting the documents stored in the JSON columns to an internal binary format. That optimized format allows quick read access to document elements and look up sub-objects or nested values directly by key or array index, without reading all values, before or after them, in the document.

Data Storage in the IoT Context

Several works addressed the database problematics in the IoE context. For instance, Bell (2016) presents a full manual to prepare an IoT system, from the building of hardware devices for data acquisition to the data storage in a relational database (MySQL). A comparison between MySQL and a MongoDB instance for IoT application is made by Rautmae & Bhalerao (2016). The authors use a simplified relational
Application of Machine Learning Algorithms to the IoE

Figure 3. Comparing relational and document-oriented databases structures

Table 1. Comparing (common) relational and MongoDB terminology

<table>
<thead>
<tr>
<th>Relational Databases</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID Transactions</td>
<td>ACID Transactions (on V 4.0)</td>
</tr>
<tr>
<td>Table</td>
<td>Collection</td>
</tr>
<tr>
<td>Row</td>
<td>Document</td>
</tr>
<tr>
<td>Column</td>
<td>Field</td>
</tr>
<tr>
<td>Secondary Index</td>
<td>Secondary Index</td>
</tr>
<tr>
<td>Join operations</td>
<td>Embedded documents, lookup etc.</td>
</tr>
<tr>
<td>Group by operations</td>
<td>Aggregation Pipeline</td>
</tr>
</tbody>
</table>

schema and compare the time taken to execute select and insert queries against a varying number of records and threads. Adiono et al. (2017) present a study on the selection and design of communication protocols, design of communication security, and database design for IoT based smart home systems. Applied to a smart system, the authors address the communication protocol and database design in order to be able to implement all the features of the smart home system, such as device’s categorization, scenarios, scheduling, notification etc. Kang, Park, Rhee, & Lee (2016) devised a data repository schema over MongoDB that can integrate and store heterogeneous IoT data sources, such as RFID, sensor, and GPS. Their work takes into consideration the Fosstrak, an open source RFID software platform which provides electronic product code information services implementation based on MySQL. In a related work, Kang, Park, & Youm (2016) analyze traceability requirements and an event schema for storing traceability. In (Paethong, Sato, & Namiki, 2016) is explained how to construct a database server for IoT middleware that has data distribution and low-power consumption by using credit-card size computers, guaranteeing satisfactory performances and affordable price. In this sense, they use a Raspberry Pi 2 and compare the results (operation’s time and power consumption) with the ones obtained with an x86 (namely an Intel Core i7), using MySQL and MongoDB as DBMS. The results show that the Raspberry Pi has a power consumption of about half of the x86 computer but is 80% slower than the x86.
Application of Machine Learning Algorithms to the IoE

machine. Malić, Dobrilović, & Petrov (2016) present a solution for wireless security cameras, based on IoT enabled open-source hardware and a MongoDB database as the storage system. Their system uses an Arduino Yún with a webcam to capture images which are then stored in a MongoDB database. Kiraz & Toğay (2017) compare relational (MySQL) and non-relational (MongoDB) database management systems for the storing and processing of IoT data, concluding that MongoDB has better performance than MySQL in terms of both writing and reading operations. Similar to the previous work, Seo, Lee, & Lee (2017) use MySQL and MongoDB to compare RDBMS and NoSQL in the IoT context. They gradually compare the performance of create, read, update, and delete (CRUD) operations, concluding that, in general, MongoDB is preferred over MySQL for handling large amounts of data. They also point some disadvantages on using MongoDB such as the weak transaction capabilities when compared with the relational database system and the large memory space required. Strohbach, Ziekow, Gazis, & Akiva (2015) summarize what they consider the key components required for smart city application, namely: the abundance of data sources, the infrastructure, networks, interfaces and architectures being defined in the IoT community, the vast range of big data technologies available that support the processing of large data volumes, and the ample and wide knowledge about algorithms as well as toolboxes that can be used to mine the data. In the same work, a case study from the smart grid domain that illustrates the application of big data on smart home sensor data is discussed.

Comparing MySQL and MongoDB Schemas: An Example

To better illustrate the concepts previously explained, let us design a simple schema to store data acquired from sensors into a database. Figure 4 sketches the enhanced entity-relationship (EER) model for a relational database, designed to store that sensors' data. The model includes the Sensor entity, which will be placed in a Location and read values in a certain Unit. Each sensor Reading is associated to a sensor (using the idSensor foreign key, a collection of fields in one table that uniquely identifies a row of another table or the same table) stored with a corresponding timestamp. This schema allows each location (and Unit) to have associated multiple sensors and each sensor can have multiple readings. On the other hand, each reading corresponds to a single sensor and each sensor has a single location and unit.

In MongoDB, documents are stored as BSON (usually thought as JSON documents) and a direct correspondence to the EER model in Figure 4 would translate those entities to three documents (Location, Sensor, and Reading), such as the ones presented in Listing 1.

Figure 4. Enhanced entity-relationship (EER) model designed to store data from sensors in a relational database
Application of Machine Learning Algorithms to the IoE

Listing 1. An initial model designed to store data from sensors in a JSON adequate database (e.g., MongoDB)

```json
# Location document
{
  "_id": "1801",
  "name": "Prometheus Server",
  "description": "Prometheus Server @ lab. 163 / ISE / UAlg"
}

# Sensor document
{
  "_id": "s011",
  "location_id": "507...801",
  "sensor_name": "cpu_sensor",
  "unit": "percent"
}

# Reading document
{
  "_id": "r999",
  "sensor_id": "507...011",
  "timestamp": "2018-05-18T16:00:00Z",
  "value": 10.2
}
```

Employing any of these solutions implies that, for instance, a query to obtain the readings of the cpu_sensor requires costly join operations (either directly using the SQL join operator or doing it on the application side). However, document-oriented databases, such as MongoDB, allow embedding documents. Therefore, a solution would be to embed the information in a single document as presented in Listing 2.

The use of embedded documents, and the implementation of required secondary indexes, allows reaching readings from a location without doing join operations. However, there are some limitations. For instance, justified in the MongoDB documentation, it is important to ensure that a single document cannot use excessive amount of RAM or, during transmission, excessive amount of bandwidth. Furthermore, the maximum BSON document size is 16 megabytes, which can be a serious limitation when using embedded documents. To store documents larger than the maximum size, MongoDB provides the GridFS API (see MongoDB's documentation). As an alternative, the documents could be split when they reach the size limit, after a certain number of reading, by periods of time (probably a good solution when looking for readings in time windows), or some other adequate policy. The proposed MongoDB's embed solution has other problematics, such as, the need to first retrieve the document where the insertion is to be made and do its update before uploading it again (as an alternative, see MongoDB's `$push` operator).

To avoid that insertion overhead, a solution is to do the denormalization of the data, which can be done in any of the DBMS solutions. For instance, the MongoDB’s solution could be the one presented in Listing 3, with the disadvantage of repeatedly storing the same data (e.g., location_name, description or sensor_name), which can represent a large waste of storage space and facilitate the occurrence of integrity issues.
Listing 2. An embedded document model designed to store data from sensors in a JSON liable database (e.g., MongoDB).

```json
{
    "_id": "s011",
    "location_name": "Prometheus Server",
    "description": "Prometheus Server @ lab. 163 / ISE / UA1g",
    "sensors": [
        {
            "sensor_name": "mem_sensor",
            "values": [
                {
                    "value": 35.4,
                    "timestamp": "2018-05-09T17:10:00.273Z"
                },
                {
                    "value": 35.4,
                    "timestamp": "2018-05-09T17:10:01.276Z"
                }
            ],
            "units": "percent"
        },
        {
            "sensor_name": "cpu_sensor",
            "values": [
                {
                    "value": 6.1,
                    "timestamp": "2018-05-09T17:10:00.273Z"
                },
                {
                    "value": 6.4,
                    "timestamp": "2018-05-09T17:10:01.276Z"
                }
            ],
            "units": "percent"
        }
    ]
}
```
Application of Machine Learning Algorithms to the IoE

Listing 3. Denormalized schema to store a single reading by document.

```
{ 
  "id": "s011",  
  "location_name": "Prometheus Server",  
  "description": "Prometheus Server @ lab. 163 / ISE / UAlg",  
  "sensor_name": "cpu_sensor",  
  "value": 35.4,  
  "timestamp": "2018-05-09T17:10:00.273Z",  
  "units": "percent"
}
```

MACHINE LEARNING ALGORITHMS AND APPLICATIONS

Supported in mathematical and computer science techniques, ML algorithms are mechanisms that use datasets to find patterns and correlations to build models. Those models are then applied to new data in order to predict its outcomes (Domingos, 2015; Witten et al., 2016).

ML are usually separated in two major classes, namely: supervised learning and unsupervised learning. Supervised learning has the task of inferring models/functions from labeled data, i.e., data that has an input vector and desirable target value. Those built models are then used to make predictions of the response values for new datasets. Supervised Learning is itself divided in classification and regression classes. The former one is used to make categorical predictions, i.e., to predict values where data can be separated into classes. Common classification algorithms include support vector machines (SVM) (Campbell & Ying, 2011; Suykens, Signoretto, & Argyriou, 2015), neural networks (Cartwright, 2015), decision trees (Hartshorn, 2016), logistic regression (Hosmer, Lemeshow, & Sturdivant, 2013) and k-nearest neighbor (kNN) (Shakhnarovich, Darrell, & Indyk, 2005). The latter, regression, is used to make predictions when continuous response values are desired. Common regression algorithms include linear regression, nonlinear regression, generalized linear models, decision trees, and neural networks. Some authors include other classes such as semi-supervised learning with the goal of employing a large collection of unlabeled data jointly with a few labeled examples for improving generalization performance, or reinforcement learning, which is used to support agents’ decisions based on the notion of cumulative rewards.

On the other hand, unsupervised learning is a type of machine learning where inferences are to be drawn from datasets of unlabeled data (Celebi & Aydin, 2016), automatically discovering useful patterns in such data. Several kinds of applications can be found such as pattern recognition, market basket analysis, social network analysis, information retrieval, recommender systems or fraud detection. For instance, recommendation systems are used daily by information systems to expose intelligence, making search engines, social media, e-stores, digital music services etc. Recommender system algorithms are a class of information filtering system which have the job to predict the user’s preferences, based on a given or guessed profile, i.e., companies apply learning algorithms on their huge datasets and let them oracle what customers want, instead of meticulously encoding the preferences of all consumers (Cardoso, Guerreiro, Monteiro, & Rodrigues, 2018; Rao & Rao, 2016; Ricci, Rokach, & Shapira, 2015).
As a consequence of this evolution, many agree that briefly most of the knowledge will be obtained and located in computers, i.e., as stated by Alpaydin (2016), “data starts to drive the operation; it is not the programmers anymore but the data itself that defines what to do next”. This results in entire industries building themselves around ML, along with emergent research and academic specialties.

In general, the ML work-flow is divided in four large steps: (a) get (enough) data - collect data related to the problem; (b) clean, prepare, and manipulate data – converting the data into a form that computers can operate on (e.g., convert things to numerical data and categorize data); (c) define and train the selected model using test and validation data – build a mathematical model of the data depending on the type of problem being solved (e.g., regression, classification, clustering, or recommendations); and (d) predict outcomes over new and unseen data – apply the trained model to unseen data to, depending on the problem at hand, predict values, classify or associate the data, recommend other objects, etc.

**Computational Tools**

When delving into the ML world, many times, ML users/programmers no longer must implement a large number of the necessary methods, as they are included in several, free or payed, visual and programmable frameworks. In fact, frameworks, libraries, applications, and datasets are available and deeply simplify many ML implementation by roughly providing out of the box solutions. For instance, Tensorflow is an open source software library for high performance numerical computation using data flow graphs, with strong support for ML and deep learning (Abadi et al., 2016). Its flexible numerical computation core is used across many scientific domains, being easily deployable across a variety of computational platforms (e.g., CPU – Central Processing Unit, GPU – Graphics Processing Units, and TPU – Tensor Processing Unit). Caffe2 (2018), an improvement of Caffe 1.0 (Jia et al., 2014), is a deep learning framework that provides an easy and straightforward way to experiment with deep learning. The framework, using the Caffe2’s cross-platform libraries, allows to scale applications from deployment using the power of GPUs in the cloud to mobile devices. The Amazon Machine Learning (AML, 2018) service provides visualization tools and assistants that guide all skill levels developers to use machine learning technology, allowing the creation of machine learning models without having to learn complex algorithms. The obtained models allow performing predictions in the users’ applications using an API. Another advantage is the lack of need to manage any infrastructure, being the system scalable and serving those predictions in real-time. The Azure ML Studio (Microsoft, 2018) also provides Machine Learning as a Service (MLaaS). In its most pure form, no coding is required since a visual tool provides an end-to-end flow with a drag-and-drop environment used to build, test, and deploy predictive analytics solutions on your data. The Studio includes hundreds of built-in packages and support for custom code, namely in R and Python (Elston, 2015). Furthermore, the models can be published as web services, easily consumed by custom apps or other tools such as Excel. With the objective of making ML scalable and easy, MLlib (Meng et al., 2016) provides common learning algorithms (e.g., classification, regression, clustering, and collaborative filtering), featurization (e.g., feature extraction, transformation, dimensionality reduction, and selection), pipelines, persistence (e.g., saving and loading algorithms, models, and pipelines) and other utilities (e.g., linear algebra, statistics, data handling etc.). Being part of the Apache’s Spark computing system, a unified analytics engine for large-scale data processing, MLlib provides high-level APIs in Java, Scala, Python and R. In the same context, data can be easily manipulated using Spark SQL which provides capabilities to operate on datasets and data frames with sources in data files (e.g., CSV, JSON, XML etc.), RDBMS, NoSQL DBMS etc. The Scikit-Learn
Application of Machine Learning Algorithms to the IoE

(Pedregosa et al., 2011) is an open source tools for data mining and data analysis. The library is built for the Python programming language and uses optimized libraries such as NumPy, SciPy, and matplotlib. The Scikit-Learn furnishes several ML algorithms in areas such as classification, regression, clustering, and dimension reduction. The Orange (2017) is an open-source data visualization, machine learning and data mining toolkit with a visual programming front-end. Furthermore, Orange can also be used as a Python library. WEKA (2017) is another graphical user interfaces (GUI) ML tool, written in Java and running on almost any platform. As many of the previous tools, WEKA is a collection of ML algorithms for solving data mining problems. The algorithms can either be applied directly to a dataset or called from your user’s Java code. The KNIME: Konstanz Information Miner (Berthold et al., 2009; KNIME, 2017) is a modular environment, which enables easy visual assembly and interactive execution of a data pipeline. MLJAR (2017) implements several ML algorithms including built-in hyper-parameter search, parallel training of models, automatic selection of algorithms, a web programming interface, the deploy of the fitted models in the cloud or locally, and a Python wrapper over the MLJAR API. But many other visual and non-visual libraries exist, such as, VELES (2018) (a distributed platform, which provides machine learning and data processing services for a user), the Shogun-Toolbox (Soeren Sonnenburg et al., 2017) (an open-source machine learning library that offers a wide range of efficient and unified machine learning methods), Torch (Collobert, Kavukcuoglu, & Farabet, 2011) (a scientific computing framework with wide support for ML algorithms using GPUs), or the MLPACK (Curtin et al., 2013) (a scalable machine learning library, written in C++, that aims to provide fast, extensible implementations of machine learning algorithms).

We have already referred Python and Java as languages for which several libraries and ML tools exist (Müller & Guido, 2016). However, others languages also are frequently used in ML, such as R (Elston, 2015; James, Witten, Hastie, & Tibshirani, 2013; Lesmeister, 2015; Yu-Wei, 2015), F# (Mukherjee, 2016), Matlab (Kim, 2017), or Go (Whitenack, 2017).

Machine Learning in the IoE/IoT Context

An endless number of applications and usages are expected for the data collected by the IoE devices. Control and understand of complex environments or the improvement of efficiency, accuracy, and throughput of productive activities, are just a few envisioned effects in this context. For instance, Satish, Begum, & Shameena (2017) proposed an agricultural system to monitor and scan environmental parameters and plant growth. The collected data is then utilized by pest control sensors that are capable of predicting pest behavior, in order to reduce the damage done by pests on a large scale. The data is collected making use of an Arduino together with several sensors, including air temperature and humidity (using a DHT11), soil moisture, and pH. The devices are placed in different locations and the collected information is stored on a MongoDB database. Finally, a decision tree algorithm is used to predict damages and an advisable combination of pesticides is proposed, to reduce the harm caused by excessive usage of those chemicals at a later stage. Another agricultural production system based on IoE/IoT is proposed by Lee, Hwang, & Yoe (2013). The work defines a monitoring system that examines the crop environment and provides methods to improve the efficiency of decision making by analyzing previous harvesting statistics. Among other aspects, statistical predictions, and real time and historical environmental data from IoT services are used to do growth forecast. Alam, Mehmoed, Katib, & Albshri (2016) examined the applicability of eight well-known data mining algorithms for IoT data, namely SVM, kNN, Linear Discriminant Analysis, Naïve Bayes, C4.5, C5.0, Artificial Neural Networks (ANNs),
and Deep Learning ANNs (DLANNs) algorithms. Their results on three IoT datasets show that C4.5 and C5.0 have better accuracy, are memory efficient and have relatively higher processing speeds, while ANNs and DLANNs can provide highly accurate results but are computationally expensive. Ahmed et al. (2017) studied the role of big data in IoT, discussing the big data processing and platforms, key requirements for big data processing, and analytics in an IoT environment. They also present a taxonomy of big data and analytics solutions that are designed for IoT systems, categorized as big data sources (e.g., city management, manufacturing), system components (e.g., data acquisition, data retention), big data enabling technologies (e.g., ML, commodity sensors), functional elements (e.g., data input, data output), and analytics type (e.g., descriptive, predictive or prescriptive). A complex of correlation-based methods for security incidents detection and the investigation in large-scale networks of heterogeneous devices such as IoT is proposed by Lavrova & Pechenkin (2015). The proposed approach was inspired by the SIEM (Security Information and Event Management) systems, which deploy multiple groups of agents to collect security related events from end-user devices, servers, network equipment, firewalls, antivirus, intrusion prevention systems etc. The decision about the existence of a security incident is supported in results of event correlation, by detecting interconnections between events from different devices. The authors also state that the complexity of security incidents detection in the IoT, among other things, is derived from the high heterogeneity of “things” and the low capacity of many of those “things”, which does not allow integration of more complex protection means. Shanthamallu, Spanias, Tepedelenlioglu, & Stanley (2017) made a survey of ML methods, analyzing several of those methods (e.g., linear regression, logistic regression, SVM, k-NN, and deep learning) and many application in the IoT field. Another survey on data mining for the IoT was made by Chen et al. (2015). The work starts by doing an analysis on the data mining functionalities, followed by a brief analysis of several ML methods for classification, clustering, association analysis, and time series analysis, and finishes with a study on the use of data mining in several contexts (e.g., e-commerce, industry, health care, and governance) associated with IoE/IoT.

Building an ML/IoT System in Three Steps

This section proposes a simple example of ML/IoT system. As always, to implement a ML system, data is needed. So, let us start by implementing a web service which allows the collection of data from any device connected to the Internet, i.e., to receive data and store it on a database (a MongoDB database on this example). Stored data can then be accessed directly the applications (see an example below) or using the web service (not implemented in the presented example).

A solution to implement a web service is to use the Python programming language and, in particular, the Flask micro web framework (Grinberg, 2018). Listing 4 presents a snippet of the webservice’s basic code. The route decorator is used to bind a function to an URL, in this case the new_reading function to the http://SERVER_IP:5000/iot/api/v1.0/reading URL, where the webservice is located at SERVER_IP IP address and 5000 is the Flask’s default port.

We should notice that the webservice and MongoDB database are presumed to be running on the same location and data posted to the server in JSON format will be inserted into the MongoDB without any kind of further validation. Out of the scope of this chapter, it is recommendable to read more about several issues (e.g., security) in the Flask’s documentation or in (Grinberg, 2018).
Application of Machine Learning Algorithms to the IoE

Listing 4. Example of web service implemented in the Python's Flask framework.

```python
from flask import Flask, request, abort, jsonify
from pymongo import MongoClient

# create a connection to a MongoDB server running in your localhost
client = MongoClient('localhost', 27017)
# Set 'db' to use the IoT database
db = client.iot

# create an instance of the Flask class -- see documentation
app = Flask(__name__)

@app.route('/iot/api/v1.0/reading', methods=['POST'])
def new_reading():
    # check if a valid json was posted
    if not request.json:
        abort(400)

    # insert data into the 'readings' collection
    db.readings.insert_one(request.json)

    # reply with an 'ok'/201 message
    return jsonify({'status': 'ok'}), 201

if __name__ == '__main__':
    # start the application in debug mode
    app.run(debug=True)
```

Two examples of how to post data to the webservice are given next. The first example gets the utilization percentages for the user’s CPU time from a computer using Python and makes the posting of this data to the above implemented server (Listing 5). In this case, a JSON is posted to the webservice running on the server located at SERVER_IP IP(port 5000). Data includes information about the sensor’s name, the acquired value (in this case, a float representing the current system-wide CPU utilization by normal processes executing in user mode, as a percentage), a timestamp, and the unit type of the posted value.

An environmental IoT device is proposed in the second example. This IoT sensor node can measure and transmit the temperature and humidity of a location to the Internet based webservice. The diagram of the projected IoT device is presented in Figure 5, being composed of an ESP8266 development board (NodeMCU) which is connected with a DHT22 (temperature and humidity) sensor. The ESP8266 microcontroller supports IEEE 802.11 communications and thus can be connected to any Wi-Fi network. Furthermore, the ESP8266 can be programmed in languages such as C++ or Python, and the microcontroller’s programming framework includes several libraries, which support standard communication protocols like HTTP, TCP and IP (among many others).

Listing 6 presents an example of a Python code to run on the device that will to post JSON formatted data, similar to the ones in Listing 3, to the API’s endpoint located at http://SERVER_IP:PORT/iot/api/v1.0/reading where, as before, SERVER_IP:PORT are the webservice’s IP and corresponding port, WIFI_SSID is the wireless network service set identifier (SSID) and WIFI_PASSWORD the corresponding authentication password. The temperature and humidity readings are sent every 30 seconds (30000 milliseconds) and the device’s time is reset every one hour (3600000 milliseconds).
Listing 5. Example of the posting of CPU usage data to the webservice

```python
import datetime, psutil, requests

# run forever
while True:
    response = requests.post('http://SERVER_IP:5000/iot/api/v1.0/reading',
        json={
            'location_name': 'Prometheus Server',
            'sensor_name': 'cpu_user_sensor',
            'value': psutil.cpu_times_percent(interval=1).user,
            'timestamp': str(datetime.datetime.utcnow()),
            'units': 'percent'
        })
```

Figure 5. Temperature and Humidity sensor connected with an ESP8266 development board

Finally, some ML can be done using the stored data. The code snippet in Listing 7 shows a basic script to train a model to predict the temperature given the humidity, month, day, hour, and minute. The proposed example starts by getting the data from the MongoDB database, filtering it to contain data from the sensor with location_name equal to “lab. 163 @ ise.ualg” and sensor_name equal to “Environment 1”. Pandas library (McKinney, 2011) is used to do some data manipulation and then Scikit-learn package (Pedregosa et al., 2011) is used to split the dataset into training and test data, do a grid search
Application of Machine Learning Algorithms to the IoE

Listing 6. Python code used to program the IoT device in order to send the readings to the information system.

```python
import dht, machine, network, ntptime, urequests

def read_send(timer_read_send):
    """Post collected data to the webservice"""
    sensor.measure()
    datetime = rtc.datetime()
    urequests.post('http://SERVER_IP:5000/iot/api/v1.0/reading',
                   json={"location_name": "lab. 163 @ ise.ualg",
                         "sensor_name": "Environment 1",
                         "temperature": {},
                         "humidity": {},
                         "timestamp": '{}/:02d}/:02d}:{:02d}:{:02d}'
                   .format(sensor.temperature(), sensor.humidity(),
                           datetime[0], datetime[1], datetime[2],
                           datetime[4], datetime[5], datetime[6])

    def update_rtc(timer_update_rtc):
        """set device’s time"""
        ntptime.settime()
        wlan = network.WLAN(network.STA_IF)
        wlan.connect(WIFI_SSID, WIFI_PASSWD)
        while not wlan.isconnected():
            pass
        rtc = machine.RTC()
        ntptime.settime()
        sensor = dht.DHT22(machine.Pin(5))
        timer_read_send = machine.Timer(1).init(
            period=30000, mode=machine.Timer.PERIODIC, callback=read_send)
        timer_update_rtc = machine.Timer(2).init(
            period=3600000, mode=machine.Timer.PERIODIC, callback=update_rtc)

with cross validation to find a good set of parameters for a Support Vector Regression (SVR) algorithm (Basak, Pal, & Patranabis, 2007), and store the model for future use.

The grid search with cross validation phase can be very time-consuming depending, for instance, on the number of observation feed to the model and the number of tested parameters. A solution might be the pruning of the number of observations using the PyMongo’s limit method to specify the maximum number of documents the cursor will return (in the example, this number was limited to 10000). In this case, it might also be advisable to sort and filter the observation taking into consideration their timestamp. It is a good idea to observe the score value over the test set, which returns the coefficient of determination R^2 of the prediction, being the best possible score 1.

Finally, Listing 8 shows a snippet of code to predict the temperature for a humidity of 89% on a 5th of January, at midnight. The previously saved model is loaded and the predict method does the estimate.
Listing 7. Example of the training of a Support Vector Regression (SVR) model to predict temperatures given humidity, month, day, hour and minute.

```python
from sklearn.externals import joblib
from pymongo import MongoClient
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

if __name__ == '__main__':
    # get data from the 'readings' collection into a pandas' dataframe - limited to 10000 observations
    readings_collection = MongoClient().IoT.readings
    df = pd.DataFrame(list(readings_collection.find({"sensor_name": 'Environment 1', 'location_name': 'lab. 163 @ ise.ualg'}, {"_id": False, 'temperature': True, 'humidity': True, 'timestamp': True}).limit(10000)))
    # do necessary castings
    df["humidity"] = df["humidity"].astype(float)
    df["temperature"] = df["temperature"].astype(float)
    df["timestamp"] = pd.to_datetime(df["timestamp"])  # split the dates into its components
    df["Y"] = df["timestamp"].dt.strftime('%y')
    df["M"] = df["timestamp"].dt.strftime('%m')
    df["D"] = df["timestamp"].dt.strftime('%d')
    df["H"] = df["timestamp"].dt.strftime('%H')
    df["Mi"] = df["timestamp"].dt.strftime('%M')
    # get an array of observation (X) and corresponding targets (y)
    X = df["humidity", "M", "D", "H", "Mi"].values
    y = df["temperature"].values
    # split the data set into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
    # train an SVR model, using grid search and cross-validation - see GridSearchCV and SVR documentation - this step takes some time!
    param_grid = {'degree': [2, 3], 'kernel':['poly', 'rbf'], 'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
    model = GridSearchCV(estimator=SVR(), param_grid=param_grid, cv=5, return_train_score=True, verbose=10, n_jobs=6)
    model.fit(X_train, y_train)
    print("'best score': {} \n best params: {} \n score over test: {} \n Best estimator: {}'\n'.format(model.best_score_, model.best_params_, model.score(X_test, y_test), model.best_estimator_))
    # store the model for future usage
    joblib.dump(model, "model.joblib")
```

**Application of Machine Learning Algorithms to the IoE**

**Listing 8. Load stored model and make a temperature prediction**

```python
from sklearn.externals import joblib
# load stored model
model = joblib.load('model.joblib')
# predict the temperature
print(model.predict([[.89, 1, 5, 0, 0]]))
```

**SOLUTIONS, RECOMMENDATIONS, AND FUTURE RESEARCH DIRECTIONS**

Within such a vast research field there is no single set of solutions, no definite recommendation. One thing is for sure: IoT, IoT and ML are here to stay. In such a massive environment, vertical specialization will be an asset as a “Jack of all trades, master of none” will not properly tackle the nuclear issues. Certainly, a general knowledge of all fields will be of interest, as it will allow the understanding of intrinsic limitations, enabling properly designed architectures from a macro perspective. This assumption also results from the observation of the huge number of research and development fields involved, which span far beyond algorithms and software to include sensors, tags and actuators, communications protocols, power feeding (e.g., consumption, use of renewal sources, batteries) etc. All these fields are passible of broad investigation, as easily found in many recent documents. Nevertheless, soft development, the main focus of this chapter, has also many things to improve. The longevity, the amount, and the type of generated data will require new storage techniques and new algorithms. On the storage side, deep research should be made in distributed databases, capable of locally storing and efficiently retrieving data on request. These distributed databases must be adequate to the data’s type and size, and especially ready to synchronization, in some cases a major issue as data might be outdated in very short intervals of time. New ML algorithmic solutions, capable of reaching better accuracies but also in a more efficient way, are also a major research field. Once again, algorithms capable of returning answers in real-time, possibly in computational constrained environments, will certainly be of major interest. Therefore, mathematical optimization, new data structures, and new ML methods which take advantage of the particularities of the data, expected outcomes, and computational restrictions are of high interest. These restrictions will be even more visible when considering edge computing and edge storage as these edge environments are in general very limited on the available energy, communications and computational capacities (e.g., memory and CPU power). These limitations can make real-time solutions a possible nightmare but, when overcome, they can also significantly improve the overall performance of the systems, for instance, by doing pre-computation on the data (e.g., data aggregation or data filtering). Similarly, distributed computing on either edge nodes or on server farms will certainly be a field where improvements, possibly algorithmic dependent, will be of usage. Finally, we cannot skip cloud services as a way to use an infrastructure in a transparent manner, significantly reducing maintenance and, possibly, infrastructural costs.
CONCLUSION

Machine Learning applied to IoE is certainly more than a short-term trend. The applications to the users are steaming a flourishing field, where academies, private and public corporations, and all other entities will have key roles. The benefits will be reflected in many activities, including agriculture, financial services businesses, entertainment companies and mass media agencies, industrial manufacturers, real estate businesses, retailers, wholesalers and distributors, transportation businesses, utilities, services, among others. So, the use of the data collect by IoE devices will allow the understanding and control of complex environments around us. This understanding and control will enable better automation, better responses, and improved efficiency and accuracy. All in all, proper devices, data and ML methods will have a significant part in creating a smarter IoE.

ACKNOWLEDGMENT

This work was supported by project AGERAR (0076_AGERAR_6_E) financed by the European Union, under the scope of the FEDER program and Interreg initiative, by project M5SAR I&DT n. 3322 financed by CRESC ALGARVE2020, PORTUGAL2020 and FEDER, and by the Portuguese Foundation for Science and Technology (FCT), project LARSyS [UID/EEA/50009/2013], CIAC.

REFERENCES


Application of Machine Learning Algorithms to the IoE


Application of Machine Learning Algorithms to the IoE


James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer. doi:10.1007/978-1-4614-7138-7


Application of Machine Learning Algorithms to the IoT


Application of Machine Learning Algorithms to the IoE


ADDITIONAL READING


KEY TERMS AND DEFINITIONS

**Internet of Things (IoT):** Dynamic global network infrastructure, with self-configuring capabilities based on standard and interoperable communication protocols, where a massive number of physical and virtual things have identities, physical attributes, and virtual personalities.

**Machine Learning:** Mechanisms that use datasets to find patterns and correlations in order to build models which will be applied to new data in order to predict its outcomes.

**NoSQL Databases:** System which provides a mechanism to store data in other than tabular relations and relationships used in relational databases.

**Relational Databases:** System which provides a mechanism to store data in sets of tuples/tabular relations. Relationships between those sets are implemented through systems of primary and foreign keys.
### Table 2. Main features of the databases addressed in the chapter

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra (V 3.11.2)</td>
<td>Wide column store</td>
<td>None</td>
<td>Linux, Windows, OS X, etc</td>
<td>No</td>
<td>Yes</td>
<td>restricted</td>
<td>Own protocol, Thrift</td>
<td>No</td>
<td>Yes</td>
<td>Sharding</td>
<td>selectible replication factor</td>
<td>No</td>
</tr>
<tr>
<td>Microsoft SQL Server (V 2017)</td>
<td>Relational</td>
<td>Document store, Key-value store</td>
<td>Linux, Windows</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>SQL</td>
<td>SQL</td>
<td>Yes</td>
<td>Horizontal partitioning, Sharding</td>
<td>Yes, version dependent</td>
<td>ACID</td>
</tr>
<tr>
<td>MongoDB (V 3.6.4)</td>
<td>Document store</td>
<td>Key-value store</td>
<td>Linux, Windows, OS X, etc</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Own protocol</td>
<td>No</td>
<td>Sharding</td>
<td>Master-slave</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MySQL (V 8.0.11)</td>
<td>Relational</td>
<td>Document store, Key-value store</td>
<td>Linux, Windows, OS X, etc</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>SQL</td>
<td>SQL</td>
<td>Yes</td>
<td>Horizontal partitioning, Sharding</td>
<td>Master-master, Master-slave</td>
<td>ACID</td>
</tr>
<tr>
<td>Neo4j (V 3.3.5)</td>
<td>Graph</td>
<td>None</td>
<td>Linux, Windows, OS X, etc</td>
<td>Optional</td>
<td>Yes</td>
<td>Yes</td>
<td>Neo4j-OGM, etc</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
<td>Causal Clustering</td>
<td>ACID</td>
</tr>
<tr>
<td>PostgreSQL (V 10.3)</td>
<td>Relational</td>
<td>Document store, Key-value store</td>
<td>Linux, Windows, OS X, etc</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>SQL</td>
<td>SQL</td>
<td>Yes</td>
<td>Declarative partitioning</td>
<td>Master-slave</td>
<td>ACID</td>
</tr>
<tr>
<td>Redis KV (V 2.1.0)</td>
<td>Key-value</td>
<td>None</td>
<td>Linux, OS X</td>
<td>No</td>
<td>No</td>
<td>restricted</td>
<td>HTTP API, Erlang</td>
<td>Javascript, Erlang</td>
<td>Yes</td>
<td>Sharding</td>
<td>selectible replication factor</td>
<td>No</td>
</tr>
<tr>
<td>SQLite (V 3.23.1)</td>
<td>Relational</td>
<td>Key-value store</td>
<td>Server-less</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>SQL</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>ACID</td>
</tr>
</tbody>
</table>

(Adapted from (DB-Engines, 2018))