Journal of Internet Services and Applications

RESEARCH Open Access

A survey on data analysis on large-Scale wireless networks: online stream processing, trends, and challenges

Dianne S. V. Medeiros[1*], Helio N. Cunha Neto[1], Martin Andreoni Lopez[2], Luiz Claudio S. Magalhães[1], Natalia C. Fernandes[1], Alex B. Vieira[3], Edelberto F. Silva[3] and Diogo M. F. Mattos[1]

*Correspondence:
diannescherly@id.uff.br
[1]Universidade Federal Fluminense - UFF, Niterói, Rio de Janeiro Brazil
Full list of author information is available at the end of the article

Abstract
In this paper we focus on knowledge extraction from large-scale wireless networks through stream processing. We present the primary methods for sampling, data collection, and monitoring of wireless networks and we characterize knowledge extraction as a machine learning problem on big data stream processing. We show the main trends in big data stream processing frameworks. Additionally, we explore the data preprocessing, feature engineering, and the machine learning algorithms applied to the scenario of wireless network analytics. We address challenges and present research projects in wireless network monitoring and stream processing. Finally, future perspectives, such as deep learning and reinforcement learning in stream processing, are anticipated.

Keywords: Stream processing, Big data, Wireless, Data mining

1 Introduction

The popularization of smartphones and Internet of Things IoT devices has driven the growth of mobile data generation via wireless networks [1]. From 2012 to 2017, mobile networks have had a seventeen-fold cumulative growth, registering an increment of 71% in data traffic from 2016 to 2017 alone. Recent studies also show that 54% of the traffic generated by devices that supports cellular and Wi-Fi connectivity was offloaded via Wi-Fi in 2017 and it is expected that this number increases up to 59% by 2022 [2]. As a consequence, IEEE 802.11 networks represent the primary access network for a significant portion of end-users in various environments. In universities and businesses, most users connect to the Internet or internal services via the institutional wireless network. For instance, the *campus* Wi-Fi network of the *Universidade Federal Fluminense*[1] accounts for 547 access points, 5 Internet gateways, serving more than 60,000 users, with peak rates of 5,000 concurrently connected users, generating over 100 Mb/s per gateway of data to be analyzed. The data obtained from the monitoring and management of such a

[1]Universidade Federal Fluminense (UFF) is one of the largest universities in Brazil, considering the size of the student body.

large-scale wireless network is a rich source of knowledge about users, networking, usage, and mobility patterns [3].

Wireless network monitoring presents several challenges when compared to wired network monitoring. Using the same tried-and-true methods of the wired world, such as measuring parameters after data has passed through the wired network, does not reveal the current state of the wireless network. Those traditional methods disallow distinguishing, for example, an idle network from a network strongly congested where frames are not being delivered. Proposals to assess the network state consider active measurements but imply changes in the evaluated parameters, as the measurement changes the state of the wireless network. Indirect measurements, such as measuring channel usage through access point counters, using sensors for spectral analysis or frame capture, are employed at the cost of some loss in the accuracy of the information. On the other hand, collecting metadata from networks does not alter network state, and allows creating context-aware applications, aiding monitoring at all levels. Capturing beacons by passive scanning, coupled with information about the signal power with which they were received, allows identifying which access points are neighbors, and inferring the physical distance between them, as well as their radio coverage area. Data collection also enables basic network operations such as service charges, threat detection, isolation, and fault mitigation. Wireless mobile networks also add space-time information about users and network conditions to provide the system with end-to-end visibility and intelligence, enabling a better understanding of long-term network dynamics. Geolocation [4] or user positioning information [5] allows identifying usage patterns and detect anomalies, for example. Besides, analyzing the network-provided data enables self-coordination of network functions and network entities, allowing to build more efficient and proactive networks.

The analysis of a large volume of data from large-scale wireless networks enables identifying usage patterns, defining user profiles, detecting failures or performance drops at specific sites in the network, and optimizing channel allocation. This analysis is challenging due to the inherent characteristics of the wireless environment, such as user mobility, noise, and redundancy of the collected data. These characteristics directly impact the five fundamental dimensions of big data processing: volume, velocity, variety, value, and veracity [6]. The lack of network intelligence and fast-reaction mechanisms, often associated with the limited view inherent to flow control tools, undermine the Quality of Experience (QoE) of large-scale wireless network users and the extraction of knowledge about users and networks [7]. In this sense, the monitoring of large-scale wireless networks requires real-time processing and immediate responses to adapt the network to peak demand and sporadic user concentrations. Traditional big data processing techniques may fit this context, but for real-time network analysis, it is mandatory to employ big data streaming processing techniques. The big data streaming processing consists of handling data that is potentially unbounded in the number of samples and in the number of attributes. Samples arrive continuously and unlimited in attributes space. Hence, the universe of data attributes, as well as the statistical distribution of the attributes are unknown. The idea of stream processing is in contrast to batch processing, in which a limited and well-known datasets are processed at a turn by a massive data processing platform. Batch processing requires a large amount of memory for data storage and implies a higher latency in generating processing responses. Streaming data processing, in turn, incurs lower processing latency of each data sample and imposes no memory constraints on the storage of incoming data.

Streaming data processing requires traditional machine learning algorithms to be adapted to the streaming data scenario, in which the training dataset cannot be searched due to the unlimited number of samples and the requirement of minimum latency when processing each new sample. To this end, streaming data processing platforms, such as Apache Spark Streaming [8] and Apache Flink [9], propose two distinct models of streaming data processing: micro-batch processing or sample-by-sample processing. Machine learning algorithms are subject to learning errors in response to concept drifts in streaming input data [10]. In this sense, machine learning applications should be aware of the statistical distribution of input data attributes and should check for changes in the statistics of the attributes.

The purpose of this article is to survey the main real-time streaming data processing algorithms and techniques for extracting knowledge from large-scale wireless network monitoring, the so-called Wi-Fi Analytics. The contributions of this article are the following: **(i)** present an insightful overview of wireless network monitoring; **(ii)** provide a summary of big data processing techniques and tools; **(iii)** review a theoretical overview of streaming data processing and the use of real-time training machine learning algorithms; and **(iv)** present and discuss applications in the context of Wi-Fi Analytics. The use of streaming data processing and real-time training machine learning concerns different application areas. We observe that the correlation of different scenarios could bring innovative views to the Wi-Fi Analytics applications. A lesson learned when surveying about WiFi-data's real-time management is that this area, although already massive, still has a place for innovative and disruptive new proposals, as we discuss along the next sections.

The article is organized as follows. Section 3 presents tools and methods for managing and monitoring wireless networks. Section 4 defines the steps for performing knowledge extraction on big data. Stream-processing machine learning and incremental learning algorithms are detailed in Section 5. The Apache Spark and Apache Flink streaming data processing platforms are explored in Section 6. Section 7 outlines the research challenges and future perspectives in analyzing streaming big data over wireless networks. Finally, Section 8 concludes the article. We provide a list of acronyms used throughout this paper to ease the reading, summarized following.

2 List of acronyms

3 Wireless network monitoring

Mobile Big Data (MBD) is the large data volume generated from mobile devices that cannot be processed on a single machine [11]. Figure 1 outlines the layers that compose the MBD. The first layer is the Collection Layer, in which *Sourcing* represent the variety of mobile devices that generate data using technologies such as Wi-Fi, Global Positioning System (GPS), and cellular communication. The following layer is the Preprocessing Layer, in which functionalities such as *Integration* and *Preparation* are performed. Integration is responsible for integrating data from different heterogeneous sources, being first validated, and then unified. To this end, the data is preprocessed using different techniques applied for cleansing and integration. Preparation, in turn, prepares the data for consumption in the Analysis Layer. This layer implements *Analytics* functionalities,

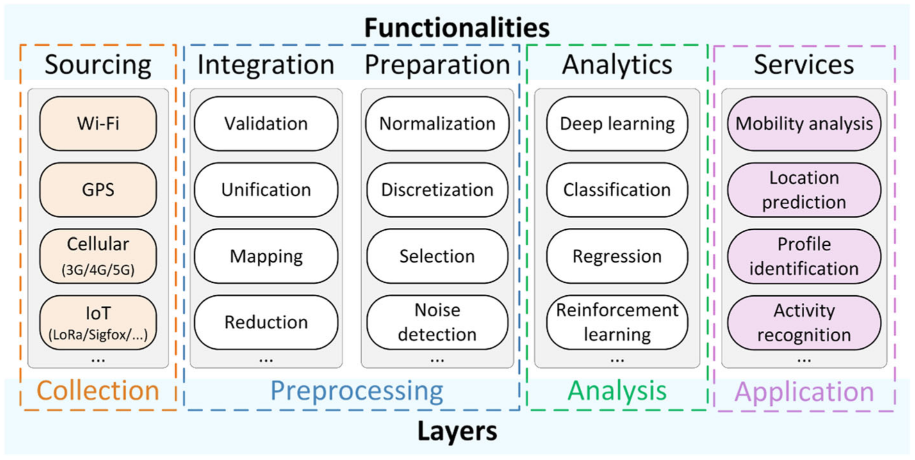


Fig. 1 The composing layers of MBD. The first layer generates the data collection. In the second layer, data is preprocessed to be consumed by the third layer, in which different algorithms are applied to analyze the data. Finally, in the fourth layer, different services use the results of the performed analyses

which are responsible for data analysis performed through different algorithms for pattern identification, classification, among others. Finally, in the Application Layer, the results produced on previous layers are used by different *Services*.

The use of machine learning techniques in wireless networks originates an entire research area, known as *Wi-Fi Analytics*. Wi-Fi analytics in the context of MDB allows characterizing IEEE 802.11 wireless networks by identifying from the collected data the actors and factors presented in the environment. The characterization, for example, can be used to propose solutions for positioning devices in indoor or outdoor environments and improve spectrum management. There are multiple approaches, such as characterization of the network topology by monitoring access points and the characterization of the spectrum. In any case, it is necessary to use specific tools to collect the data.

3.1 Knowledge-drive applications on wireless networks

The environment in which wireless networks are immersed changes constantly. The access points that build the network infrastructure are usually fixed and connected to the wired network structure. Nevertheless, user mobility causes fluctuation in the noise level to which the access points are subjected and interferes with the radio environment, as people's bodies act as a barrier to microwave propagation. In large-scale networks, new access points may be switched on and off at any time, some access points are mobile due to the proliferation of cellular Internet via Wi-Fi sharing, and the radio environment may constantly change, for example, by opening and closing doors. Thus, characterizing the network becomes a complex task and requires access points to be monitored to discover network topology, characterize the radio environment, and the users associated with the network in a quasi-real-time manner. Table 1 summarizes the main works listed on this section.

Neighborhood Inference and Characterization

A neighborhood study is useful to characterize a wireless network, as it allows the researchers to discover features related to the network itself and the radio environment. One approach to this characterization of the vicinity of an access point is to

Table 1 Summary of Knowledge-driven Applications on Wireless Networks

Reference	Research opportunity	Method used
Acer et al. [12]	Connected Objects Fingerprint	Platform to analyze Wi-Fi trace
Acer et al. [13]	Crowd Behavior	Data Analysis
Gómez et al. [14]	User Association	Wi-Fi SDN
Balbi et al. [15]	Channel Allocation	Metric-based Algorithm
Coronado et al. [16]	Channel Allocation	Wi-Fi SDN
Xu et al. [17]	Vehicles Location	Survey
Chen et al. [18]	Traffic Jam Prediction	Combine data from different wireless sources
Leung and Kim [21]	Optimal Channel Allocation	–
Maturi et al. [22]	Heuristic Channel Allocation	Dynamic channel selection
Lin et al. [23]	Channel Allocation	Interference observed between APs
Luiz et al. [20]	Channel Allocation	Interference observed between clients and APs
Shin et al. [24]	Decrease Latency during hand-off	Stores the set of channels each neighbor is operating and the set of neighboring APs on each channel
Zeljković et al. [25]	Evaluates Handover Algorithms for QoS	SDN + Machine Learning
Huang et al. [26]	Monitor wireless AP	MBD platforms, data analysis, and distributed acquisition tools
Wang et al. [28]	Optimize use of the wireless spectrum	Cooperation between access points to perform beamforming
Ghouti [29], Noulas et al. [30], Kulkarni et al. [31], Stynes et al. [32], Zhang and Dai [33], Bozkurt et al. [34]	Positioning Analysis	Machine Learning
Gonzalez et al. [35]	100.000 Traces of cellphones Dataset	–
Song et al. [36]	People's Movement Dataset	–
Toch et al. [37]	Classify user mobility applications	Clusters Similar Profiles and their future trajectory
Jiang et al. [41]	Characterization and spectrum analysis in next-generation wireless networks	Survey

collect beacons, either passively or by inducing their emission via the IEEE 802.11 probe-requests . Every access point broadcasts beacon frames periodically, usually set at one every 102.4 ms. Beacons belong to the management frames of wireless networks and include a task for various mechanisms, such as synchronization process and energy saving. Therefore, access points cannot suppress sending beacons. Capturing beacons allows the discovery of all access points in an area. Whether passive or stimulated scanning, this type of data collection only identifies access points, as stations do not emit beacons or respond to probe-requests.

The value of the received power of a beacon frame is an indicator of the radio distance between the sender and the receiver. Because the environment varies continuously, a single measurement is unreliable, but with multiple measurements, it is possible to create a reliable map of radio distances between access points. In an institutional wireless network, these measurements can be used to create channel and power configurations for each access point, aiming to minimize interference between neighboring access points, whether they belong to the managed network or a third-party network. Access points belonging to the institutional network are aware of the stations associated with them and can exchange this information with a centralized monitoring system. It allows an operator to know the number of users associated with each access point and the global number of users associated with the network. A wireless network, however, is generally unaware of the stations associated with access points from another wireless network. Knowing how many users are associated with neighboring networks requires to capture packets to identify the source MAC addresses. To infer interference caused by wireless signals from devices other than IEEE 802.11 devices, such as the interference caused by Bluetooth devices, cordless phones, or microwave ovens, mechanisms such as spectral scanning are required.

Radio Environment Network Topology and Characterization

The neighborhood study provides the discovery of the wireless topology. Such knowledge allows exploring features related to the radio environment but also provides information about users. In the context of MBD, some researches adopt Wi-Fi analytics to locate objects or users. The idea is to use management and control packets, such as probes and beacons from IEEE 802.11 to determine the location or offset of objects or users [12, 13]. It is possible to locate a user, with low accuracy, assuming he is in the vicinity of the access point to which he is associated. In general, as a station is associated with only one access point at any given time, we can use the history or sequence of access point associations to improve location accuracy or to infer the station trajectory. Physical information, such as signal strength contained in probes, can be obtained from Wi-Fi access points and used to estimate, for example, how people are moving. Nevertheless, association history has limited accuracy. The user-AP association procedure does not follow a predefined standard, but it is a vendor's choice. For the sake of simplicity, the user-AP association is usually decided according to the signal strength. This approach leads to uneven distribution of users and misuse of network resources [14]. Because the radio environment varies continuously, and the algorithms currently used for choosing access points rely on a single measurement, stations may change access points, even when static, due to interference and changes in the radio environment [15], creating a movement that does not exist. Recent proposals consider the use of Software-Defined Networking (SDN) to manage joint user association and channel assignment on Wi-Fi networks. Gómez et al. propose considering the average signal strength, channel occupation and load of each AP to optimize user association decisions on an SDN-enabled Wi-Fi network [14]. In turn, Coronado et al. also consider an SDN-enabled Wi-Fi network and present a user association scheme capable of detecting when traffic is not distributed efficiently and rescheduling user association transparently to APs and clients that are hampering network performance [16].

In the context of vehicles rather than users, location accuracy and mobility prediction from a huge data volume can be improved combining different wireless sources and technologies, such as GPS, cellular signals, such as GSM and LTE, Wi-Fi signals, among others [17]. It is also possible to determine the location and trajectories of vehicles with external data, such as maps. Thus, intelligent traffic systems using big data tools can analyze data from different wireless sources to mitigate traffic jam problems [18]. If only Wi-Fi data is available, the accuracy of the location of objects and people will depend on the density of access points on the network. The lower the density of the network, the lower the location accuracy. The collected mobility data is analyzed using classification and regression techniques to retrieve useful knowledge, such as vehicle traffic flow prediction. Among the used algorithms, we can mention time series models, Deep Learning (DL) prediction, Markov chain models and Neural Network (NN), and AutoRegressive Integrated Moving Average (ARIMA). Other approaches also propose to manage the network based on Simple Moving Average (SMA) or Simple Moving Median (SMM) indicators. These approaches calculate bandwidth and latency indicators. It is worth mentioning that, when considering latency indicators, SMM is deployed, since latency tends to suffer from the masking effect in the presence of outliers [19]. Besides, approaches based on Weihgted Moving Average (WMA) are also used to assign a greater weighting to the most recent data samples, whereas distant past samples have their weight vanished.

Large-scale wireless networks can benefit from centralized control for optimizing certain tasks, such as how to utilize spectrum and allocate users. The wireless medium can suffer interference from many sources, turning channel allocation into a challenge. The complexity of the problem grows when adding the constraint of providing acceptable performance for users subjected to channel overlap [20]. Nearby channels generate interference and decrease the performance of wireless networks in the vicinity. Therefore, the wireless network design needs to consider channel allocation from both the deployed access points and other access points already present in the area. Optimal channel allocation is an NP-hard [21] problem, but heuristics achieve good [22] channel configuration solutions.

In large-scale wireless networks, proprietary alternatives for access point management, and consequent channel assignment, perform well with a high financial cost [15]. To avoid high costs, Balbi et al. propose a channel allocation algorithm that first considers internal interference between controlled access points, and then external interference from access points with all neighboring access points. Maturi et al. propose a dynamic channel selection scheme that allows an IEEE 802.11 network to hop over the available channels, always choosing the one with the least usage [22]. The authors implement the proposal and show the feasibility of frequency hopping for IEEE 802.11 networks with industry-standard equipment. Other proposals perform channel allocation through interference observed between access points [23] and between clients and access points [20].

It is also important to consider the impact of hand-off on large-scale wireless networks, considering the quality of the experience perceived by the users. In this context, Shin et al. describe the wireless network through a graph and investigate how to decrease latency during hand-off [24]. The main idea is to develop algorithms to allow the hand-off without the station having to monitor all channels. To this end, the station stores the set of channels each neighbor is operating and the set of neighboring access points on each channel. Zeljković et al. introduce an SDN modular handover management framework, which creates, validates and evaluates handover algorithms that preserve Quality of Service (QoS) [25]. They also propose a proactive handover algorithm based on machine learning, which relies on multiple metrics to predict the future state of the network and to optimize the AP load, while preserving QoS.

Huang et al. use MBD platforms, data analysis, and distributed acquisition tools to monitor wireless access points on Unicom's 3G WCDMA network in China [26]. The authors use a data storage and analysis platform based on the Hadoop Distributed File System HDFS. Similar architecture can be employed to process data from other large-scale wireless networks, such as institutional Wi-Fi networks. Hadi et al. survey several related work [27].

Spectral Characterization

Successful frame capture depends on good channel reception and tuning. Thus, transmissions that may interfere with the wireless network, such as Bluetooth, microwave ovens, wireless telephones, and even frames that have collided or are below the sensitivity threshold given the signal-to-noise ratio, can prevent frames from being captured. Hence, spectral monitoring is an alternative to frame capture. This kind of monitoring captures all the energy that arrives at the antenna in the tuned spectrum range, regardless of the source. Low-cost spectrum capture in the frequency range utilized on Wi-Fi can be done through spectrum analyzers, e.g., WiSpy[2]. Spectrum capture provides a better view of interference sources at the monitored sites and allows defining whether a source is a transmitter from a wireless network or a Bluetooth network, for example. It is possible due to the "spectral signatures" of each technology, i.e. the format of the captured spectrum. A limitation of spectral capture is the delayed response, as identifying the "signature" requires time integration. Besides, there is a high financial cost associated with the use of spectrum analyzers, especially in large-scale networks.

Several works use spectrum characterization of an area as a tool to improve network performance. Wang et al. propose optimizing the use of the wireless spectrum through cooperation between access points to perform beamforming [28]. Access points are connected via Ethernet to make information exchange faster. The proposal is based on three pillars: (i) a cooperative scheme that allows the system to estimate phase deviations in each transmitted symbol and dynamically adjusts the phases to ensure alignment; (ii) an estimation mechanism that measures channel quality and (iii) a random algorithm of user choice to perform beam formatting with a constant computational cost. The authors show that the random algorithm can scale the network linearly and has a performance equivalent to 70% compared to more complex algorithms.

Characterization of User Mobility

The fingerprint-based positioning analysis approach is commonly used for indoor positioning and consists of two phases: one online, also known as training and one offline, also known as positioning. In the offline phase, a radio map is created using the Radio Signal Strenght Indicator (RSSI) values that are measured at existing access points in the environment. Radio maps include, in addition to RSSI values, information about the access points at which measurements were taken. In the online phase, localization is performed by combining the radio map RSSI values and the RSSI values measured by the mobile unit. It is noteworthy that physical device changes between the offline and online phases can affect positioning accuracy, as well as the choice of algorithms and their parameters in the online phase. In the positioning literature, machine learning algorithms have wide use in estimating these positions [29–33]. There are several algorithms suggested in this application, and it is a non-trivial task to find the one that best behaves in a given scenario. Comparing algorithms concerning positioning and computation time, Bokzurt et al. show that the k-NN algorithm is the most appropriate [34] when compared to Decision Tree, Naive Bayes, Bayesian Network, Sequential Minimal Optimization (SMO), Adaptive Boosting (AdaBoost), and Bagging algorithms.

[2]Available at https://www.metageek.com/products/wi-spy/

The availability of large datasets used for user location tracking has become more common with the arrival of various telecommunications-related technologies associated with MBD. Several works analyze mobility patterns using, mainly, statistical models to get the general properties of these observed patterns. For example, Gonzalez et al. analyze 100.000 traces of cellphones and identify that the distance between users follows a power-law probability distribution [35]. After identifying the main patterns of people's movement and the degree of predictability [36], researches began to focus on the development of applications based on mobility analysis. Another interesting area is based on network complexity theory and statistical tools. These studies focus on the analysis of social relations to aggregate mobility patterns in a large data volume.

Toch et al. classify user mobility characterization applications into three categories: (i) user modeling applications; (ii) locality modeling applications; and (iii) trajectory modeling applications [37]. In (i) only one user is analyzed for a period of time, aiming to predict his mobility pattern and future location. In (ii) only localities, or areas, are taken into consideration to predict the number of people passing through a certain area. In (iii) a punctual spatial-time analysis is created to identify user mobility patterns, trying to identify the users' clusters with similar profiles and their future trajectory.

MBD naturally associates with these applications, as well as the need for efficient response time processing. Thus, the modeling approach is of paramount importance. Different approaches use different methods, which directly influence the outcome. Most methods are based on machine learning, whether supervised or not. Other methods use nonlinear time series analysis, Markovian models, or regression [33, 38].

Next-Generation Networks

Next-generation wireless networks, such as 5G, must support extremely high data rates and new and diverse application paradigms that will require new wireless radio technologies [39]. The challenge is to help wireless networks to learn the decision-making process adaptively and intelligently so that the diverse requirements of such networks are met. In this context, machine learning is one of the most promising artificial intelligence paradigm, designed to support intelligent radio terminals [40]. Future 5G-enabled smart mobile terminals are expected to autonomously access the best spectral bandwidths with the aid of learning. Mobile terminals are also expected to perform sophisticated spectral efficiency inference, controlling transmitting power, energy efficiency adjustment, and transmission protocols based on learning and inference of QoS. In this sense, Jiang et al. classify various topics and works related to machine learning applied to the characterization and spectrum analysis in next-generation wireless networks [41]. The paper classifies the techniques in supervised, unsupervised, and reinforcement learning. It is noteworthy that, in the context of spectrum characterization, several distinct techniques can be applied, highlighting the supervised learning algorithms based on Bayesian networks, regression models, K-Nearest Neighbor (KNN), and Support Vector Machine (SVM).

3.2 Main wireless network data crawling and analysis tools

Network traffic monitoring classifies into two categories: active and passive. Active approaches, implemented by common tools, such as *ping* and *traceroute*, inject traffic or probes into the network and observe the results, such as loss and round trip time. Passive approaches, on the other hand, do not directly interfere with existing network traffic. Passive tools observe the current network traffic to infer its properties. The passive approach may provide more information about the network traffic. Indeed, as tools can collect all network packets, they can perform analysis on the whole traffic [42]. The collecting task, however, can be costly. In this sense, exporting only the flow information is a more scalable variation of the passive monitoring approach which makes it suitable for the use in high-speed networks. To capture and export flow information, a tool aggregates network packets into flows and the correspondent aggregated data are exported for future analysis. A flow, in this case, is defined as a set of packets passing through an observation point on the network over a certain period of time, such that all packets have a set of common properties [42]. These common properties may include packet header fields such as IP source and destination, port numbers, transport protocol type, and etc. In this article, we discuss some of the most important monitoring protocols, such as SNMP, NetFlow, sFlow, IPFIX, and OpenFlow, presenting their main features, limitations and use in wireless networks, as summarized in Table 2.

Table 2 Comparison among main wireless network monitoring tools

Monitoring Tool	Strengths	Weaknesses	Main features
SNMP	- Simple usage	- Unsuitable for large networks	Collect information about device and network states described in MIBs using a polling mechanism
		- Large delays due to polling	
		- No authentication	
		- High computational demand	
NetFlow	- Flexible	- Overload in routers and switches	Protocol for flow analysis that collects and aggregate information in a central server
	- Granular and agreggated data collection and analysis	- Limited visibility of traffic	
	- Use of templates	- Proprietary protocol	
sFlow	- Open protocol incorporated in devices of different manufacturers	- Messages without encription	Protocol for flow analysis through random sampling of data packets
	- Flexible	- Less packet details when compared to NetFlow	
	- Low computational demand	- Accuracy issues when high sampling is required	
IPFIX	- Flexible	- Complex multistage process in templating	Evolution of NetFlow which is able to export traffic information from Layer 2 to Layer 7
	- Variable lenght fields	- Delays due to miss template packets	
	- Data types defined by the user		
	- Multi-layer packet monitoring		
OpenFlow	- Open protocol		
	- Flexible	- Authentication issues	Real-time traffic control and monitoring
	- Multi-layer forwarding and monitoring table	- Not available in many commercial network devices	
	- Easy adaptation to new network requirements		
INT	- Based on P4	- High network overhead	Reports of network state via the data plane

Simple Network Management Protocol (SNMP) is a management protocol in the application layer from the TCP/IP stack that uses the UDP transport protocol services to send and receive messages across the network [43]. In short, the protocol gets information and data from SNMP servers through requests from a manager to an SNMP agent, which is a special software installed on monitored network devices. Each network element has variables that represent its information and current state. The manager accesses information and even change a given element property. In this sense, each managed network element must have an SNMP agent and a Management Information Base (MIB). The MIB contains information about managed objects, which are real-world abstractions, representing the system resources that are queried or changed. The managed objects have query and change permissions such as read/write permissions. Reading the current object value is equivalent to reading its current state, and writing in object reflects in its state. For example, SNMP accesses (read) settings from an access point, such as active interfaces, channels in use, and power. Currently, there are three main versions of the protocol (SNMPv1, SNMPv2, and SNMPv3). SNMP has some well-known limitations. The protocol, for example, is not suitable for managing very large networks. Since it is based on a polling mechanism, managing many network elements can cause an excessive delay. In addition, basic SNMP setup has no authentication mechanisms. Finally, it is highlighted that SNMP demands high computational resources so that applying it to wireless networks can deplete the resources of the network elements. In this sense, SNMP is not suitable for wireless sensor networks or networks with limited resource devices such as can be found in IoT deployments. Nevertheless, SNMP is still a feasible and widely used monitoring protocol mostly available on commercial products. Widely deployed network monitoring tools, such as Nagios[3] and Zabbix[4], rely on SNMP for collecting statistics about network assets.

NetFlow is a Cisco proprietary protocol used for flow analysis. It was implemented circa 1996 and allows the collection and aggregation of network traffic information on network elements compatible with the protocol [42, 44]. NetFlow collects the flow information and sends its records to a centralized location on the network. In this centralized point, data are analyzed by a network administrator who can determine, among others, the source and destination of traffic, the classes of service traffic on the network and the causes of possible network congestion. In this sense, examples of the typical use of statistics collected by the NetFlow protocol are: (i) monitoring the bandwidth usage of links; (ii) detection of threats in networks, such as Denial-of-Service (DoS) attacks; (iii) record of use and accounting of network resources; (iv) investigation of the causes of congestion and performance degradation of network resources and applications. Figure 2 shows a typical monitoring configuration using NetFlow, which consists of three main components [42]. The *NetFlow Exporter* aggregates packets into flows and exports flow records to one or more flow collectors. The aggregation is done based on the source/destination IP addresses, source/destination ports, classes of service, IP protocol, and source interface. The *NetFlow Collector* is responsible for receiving, storing, and preprocessing flow data received from a flow exporter. Finally, the *Analyzer* analyzes the received stream data, which covers specific contexts, such as intrusion detection and traffic profiling.

[3]Available at https://www.nagios.com/.
[4]Available at https://www.zabbix.com/.

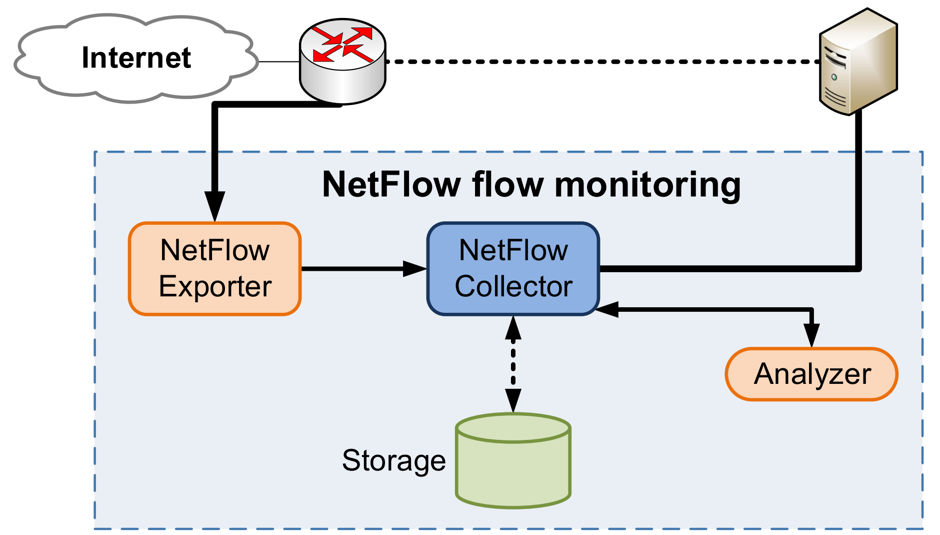


Fig. 2 A typical flow monitoring setup using NetFlow. NetFlow-enabled devices, NetFlow exporters, create records by aggregating packets into flows and exporting the records to flow collectors. NetFlow collectors store and preprocess the received data and forward it to the data consuming analyzer to generate useful knowledge

Each packet forwarded in a network element is examined and the first packet that fits any rule triggers the creation of a record in the NetFlow cache. The packet is then forwarded, and other packets that follow the same rule are added to the newly created flow record. The counters in that record are updated with each new packet that matches the rule pattern. The cached flow information is exported to the NetFlow collector which, in turn, stores the stream data on a regular database. The NetFlow protocol uses UDP or SCTP to transmit the records to the NetFlow collectors. Finally, Analysis Applications analyze flows. These applications may analyze the received flow focusing on intrusion detection or traffic profiling. The applications are also responsible for presenting the data and creating reports.

One of the main weaknesses related to NetFlow refers to the overhead it imposes on the network infrastructure. Collecting and sending flows add overhead to routers and switches, which are often already overloaded. The more details administrators insert into a single flow tuple rule, the more overhead. In many cases, network administrators disable the protocol to improve network performance. It is noteworthy that NetFlow has limited visibility of the traffic, or packets, forwarded. As a consequence, collateral communication (LAN/VLAN)[5] is not counted in the flows exported.

The **sFlow**, as well as Juniper (jFlow), Ericsson (rFlow), and Huawei (NetStream) follow the same NetFlow approach. The technology sFlow[6] became an industry-standard defined in the RFC 3176 [45]. The technology uses random sampling of data packets. The sFlow agent is incorporated into switches and routers from different manufacturers. The agent is a software process that associates interface counters and flow samples, generating sFlow datagrams that are immediately sent to sFlow collectors via UDP datagrams. The sFlow datagrams contain information about the sFlow version, agent address, source IP,

[5]Collateral communication refers to flows that are forwarded between nodes in the Local Area Network and are not addressed to the NetFlow-enabled gateway.
[6]Collectors for sFlow are available at https://sflow.org/products/collectors.php

sequence number, number of samples and, normally, up to 10 flow samples. The immediate data sending minimizes memory and processing usage. Packets are typically sampled using Application-Specific Circuits (ASIC) to ensure high wire-speed performance. The sFlow data contains the complete packet header and routing information. The sFlow is capable of running in Layer 2 and capturing non-IP traffic.

Internet Protocol Flow Information Export (IPFIX) is also an alternative tool for NetFlow [42, 46, 47]. In sum, it is a standard derived from NetFlow v9 that is capable of exporting any traffic information from Layer 2 to Layer 7 to the flow collector. It is a flexible protocol that supports variable length fields and allows the collection of information such as URL or host address, as well as the types of data defined by the user. More in deep, previous versions of NetFlow (e.g., NetFlow v5) were rigid, lacking additional data types (e.g., IPv6, MAC addresses, VLAN, and MPLS). Cisco introduced the idea of templates to relax the flow monitor. Templates provide a less rigid basis for a collector to interpret flow records. Templates make newer versions of NetFlow (e.g., NetFlow v9) more flexible. However, while monitoring a flow, each template receives a template ID. A given template ID might be used by multiple network equipment vendors, in which case each vendor's equipment would likely be crawling a different set of flows. IPFIX allows networking hardware vendors to specify a Vendor ID to create their proprietary information to be exported, avoiding namespace conflicts.

Despite the advantages, IPFIX presents some cons, mostly because it is based on templates. First, templates packets are infrequent. It should be noted that, until one gets a template packet, it is not possible to decide what the collected flow data means. As a consequence, in case of a crawling process miss template packets, the flow exporting process may considerably delay. The IPFIX protocol specifies that it is not supposed to cache templates across multiple exporting devices. The result is that figuring out which types of flow data the collector is collecting can be a slow process. Finally, templating involves a complex multistage process.

OpenFlow is a network standard defined by the Open Networking Foundation (ONF)[7] for implementing SDN in networking equipment. The protocol defines the communication between an OpenFlow switch and an OpenFlow controller, allowing the controller to program the switch to handle incoming traffic. The communication between the controller and the switch happens through communication channels. For each packet from the user traffic that arrives at an OpenFlow switch, the switch checks a flow table searching for a matching entry. An entry in the flow table has packet match fields, priority, counters, packet processing instructions, timeout, cookie, and flags. If a matching entry exists, the packet is processed and forwarded according to the entry. If the packet matches only the "table-miss entry", it is forwarded to the controller for further action. The controller, in turn, must choose to either drop the packet or create a new entry in the flow table for this new flow. If no matching entry exists, the switch drops the packet [48, 49]. Such a protocol can be used in wireless local area networks based on SDN to monitor network traffic [7].

OpenFlow-based networks benefit from more flexibility, agility and facilitated adaptation to modification of requirements. Nevertheless, one of the major drawbacks of OpenFlow protocol is its vulnerability due to the authentication mechanisms between

[7]ONF is a non-profit operator led consortium driving transformation of network infrastructure and carrier business models, its website is available on https://www.opennetworking.org/.

controllers and switches. Such vulnerability allows an attacker to cause a DoS attack against the controller because the controller cannot verify the switch identifier, which can be spoofed, involved in the OpenFlow handshake [50].

In-band Network Telemetry (INT) is a network monitoring framework that allows collecting and reporting the network state via the data plane, independently of the control plane. It is based on P4[8] and can be executed on a variety of programmable network devices [51]. Each packet has header fields that are interpreted as telemetry instructions by network devices. As such, data packets can query the internal state of a switch, e.g., queue size, link utilization, and queuing latency. The response for each query is written into the packet as it transits the network. There are three types of nodes that participate in the INT: traffic sources, traffic sinks, and transit hops. The source is a trusted entity capable of creating and inserting INT headers with INT instructions into the packets it sends. The INT instructions are included in the INT header as a list of metadata to be monitored at each hop. Transit hops are networking devices that add metadata to the INT packet according to the INT instructions in the packet header[9]. The sink is a trusted entity that extracts the INT headers and collects the path state added to the INT headers. Then, it forwards the original data packet to the destination node and the header with the collected metadata to a monitoring engine. The monitoring engine is responsible for processing the metadata to obtain the real-time network information [52]. The existing INT framework attaches the INT header and metadata stacks to all data packets, which can result in network overhead, especially for small size packets; and overload of the monitoring engine for flows with high packet arrival rates, increasing the processing latency [52].

4 Big data processing

As discussed in the previous section, large-scale wireless network monitoring generates larges amounts of data, which characterize the big data environment. Processing big data is commonly associated with machine learning applications. Machine learning refers to the ability of computers to learn without being explicitly programmed [53] and it is based on probability theory and statistics, Bayes theorem, and optimization, being the basis for analysis and data science in the big data scenario [54]. Transforming data into information and useful knowledge through machine learning techniques is the most important step in any big data analytics [55]. Data analytics are performed either in batch (fixed amount of data, no real time) or stream (continuous data, real time). Before submitting data for analysis, it is necessary to prepare it. Thus, an essential step before analysis is data preprocessing.

4.1 Batch data analysis

Batch processing is the most traditional method of data processing. It relies on databases that permanently register the information that is processed later through queries. In batch processing, it is also common to use Online Analytical Processing (OLAP), data mining techniques [9], and distributed processing with the aid of tools, such as Hadoop. Traditional data processing performs operations as relational selection, projection, and

[8]P4 is a language for programming the data plane of network devices, specifications available on https://p4.org/.
[9]INT complete specification is available on https://p4.org/assets/INT-current-spec.pdf

aggregation, among others. After collection, data are stored in disk and later processed, as there is no need for real-time response.

Batch data processing is efficient to process large volumes of data when transactions are collected over a spaced period. Thus, the data is collected, stored, processed and, finally, the results are generated. The architecture and implementation of traditional databases used in batch processing are not designed for fast and continuous data input and reading [56]. This is not an issue, however, since batch processing techniques hold flexible requirements regarding the throughput of processing results, batch processing unpairs the real-time knowledge extraction. In the context of wireless networks control and monitoring, batch processing is useful because archived data is processed to generate information to be used later to enhance the network.

4.2 Real-time streaming data analysis

Applications with strict response-time requirements, whether they are executed by an operator or automatically, need to combine streaming data collection with inference and correlation techniques. Due to the high arrival rate of streaming data and the high volume of data on a large network, traditional processing algorithms lack responsiveness to iterate through the data repeatedly. In these scenarios, one-pass algorithms that use small amounts of memory are required, characterizing real-time stream data analysis. The analysis assumes that the data volume is not static but incremental, forming a data stream. Examples of data streams include web server text, application logs, sensing data, and state changes in transactional databases [9]. In this sense, the use of traditional batch processing becomes unfeasible, since processing response needs to respect a very small time window. Strategies that group data by hours, days, or weeks generate an intrinsic delay that renders some applications unfeasible.

Although much of the useful information to organizations with large databases may be obtained through batch processing, some specific applications require real-time results from the analysis of the collected data. It allows the organization to take immediate action in situations where a very short delay is sufficient to cause problems. For example, performing an intrusion detection analysis requires a rapid response so that the intrusion avoids major losses to a machine or network. Thus, the main goal of real-time stream processing is to extract information for decision making in a very short time window. By performing real-time streaming data analysis, processing and memory resources become a potential bottleneck, especially in the context of Big Data. Data arrives in large volumes and may arrive in bursts, and results should be presented in real or near real-time. Thus, the use of secondary memory (disk) is generally not possible in these scenarios [57], as writing and reading times are high and incompatible with the application's time constraints. Real-time stream processing uses the "compute-first, store-second" scheme. The need for a very fast response regarding a large volume of data usually requires the use of distributed systems for big data processing.

Other characteristics are also important for choosing stream processing instead of batch processing [56]: (i) when elements arrive inline, because they should be processed as soon as they are received; (ii) when the processing system has no control over the order in which the elements arrive; (iii) when the amount of data is unknown, as streams have no restrictions on size or duration; and (iv) when processed elements are usually discarded or archived and, therefore, not easily accessible.

Data streams are represented through different models, as follows:

- **Time Series Model.** A time series is a set of data samples made at constant time intervals over a time window. Given a sample of a time series, it can be divided into three basic components, namely: trend, related to a long-term view; seasonality, related to systematic calendar movements; and irregularity, which is related to non-systematic short-term fluctuations. In a university's wireless network, for example, the number of users on an access point often reveals seasonality, as the number of users varies by day and night, weekends, and vacation periods. Usually, a data analysis performs a seasonal adjustment, identifying and removing systematic and calendar-related influences. Periods without users due to the calendar should be removed in the analysis of the average network load for sizing and positioning of access points. The seasonal adjustment process is important because seasonal effects can mask the true movement of the time series as well as non-seasonal features that are of interest.
- **Cash Register Model.** To define this model, it is assumed that the input signal is represented by a stream $a_1, a_2, \ldots, a_n$, with sequential arrival and a signal A, defined by the function $A[j] : [1 \ldots N] \rightarrow R$. The input samples a_i are increments for the signal $A[j]$. For example, let be $a_i = (j, I_i)$, $I_i > 0$, and $A[j]_i = A_{i-1} + I_i$. In this case, A is a flow in the cash register model, where $A[j]_i$ is the signal state after the arrival of the i-th sample [58]. This model is one of the most popular. It can represent, for example, the total access time of each user on a network after multiple accesses, where j represents each user, I_i is the time of each access of the user, and $A[j]_i$ represents the total access time of user j until the moment i. Another example is monitoring MAC addresses that connect to an access point, where each MAC receives a different value for j and $A[j]_i$ represents the total number of accesses from that MAC.
- **Turnstile Model.** This model is more generic than the previous ones but is similar to the cash register. In the turnstile model, however, I_i can assume positive or negative values. This is the most appropriate model for studying broadly dynamic situations, where elements can be inserted or removed [58].

4.3 Data preprocessing techniques

Raw data collected from numerous heterogeneous sources comprise a large amount of worthless or unnecessary information. It results in datasets with different quality levels, which vary according to their completeness and quantity of redundancy, noise, and inconsistencies [55]. Raw data, also known as primary data [59], are of poor quality and when directly processed result in the use of large storage space and generate poor quality information. Thus, data quality directly impacts the performance of the employed processing algorithms and the quality of the obtained results. Thus, one of the first steps to which the collected data is submitted before analysis is preprocessing. The purpose of preprocessing is to increase the quality of the dataset and possibly reduce its volume by cleaning up inconsistencies and noise, eliminating redundancy and integrating data to provide a unified view. This step occurs either before or after data transmission, but it is best suited prior to data transmission due to bandwidth and storage space requirements [55].

After preprocessing, the obtained dataset is considerably reliable and suitable for the application of data mining algorithms [59]. Data preprocessing is of particular importance for large wireless networks due to a large amount of collected data and the redundancy therein. For example, two distinct access points may collect data about the network in a region where there is coverage overlap. The data collected by both access points for this common region is strongly redundant and does not need to be stored completely. Moreover, if the data is processed in the presence of redundancy, the conclusions tend to be biased.

Preprocessing includes several techniques to promote data *cleansing*, *integration*, *shrinking*, and *transforming* [55, 60]. **Data cleansing** techniques determine the accuracy and completeness of the data, fixing any problem by removing or adding data to the original dataset, and resolving inconsistencies. Actual datasets are often incomplete due to failures in the data collection process, limitations in the data acquisition process, or cost constraints, which prevent the storage of some values that should be present in the dataset [60]. Also, data errors need to be addressed by documenting both occurrences and error types so that procedures can be modified to reduce future errors [55].

The first step of the cleansing search for outliers using various commercial scrubbing and auditing tools. Data scrubbing tools use simple knowledge of data mastery to detect and correct errors by employing parsing and fuzzy matching techniques. Data auditing tools[10]−[14], some of which have more than just data auditing capabilities, discover rules and relationships, identifying data that violates the conditions encountered. Thus, they employ, for example, statistical analysis to identify correlations, and grouping algorithms to identify outliers.

When identifying missing data, the trivial solution is to drop the sample, but it may result in misrepresentation of the learning process and discarding important information. Another approach is to manually complete the data, which may be impractical. A global constant may also be used to complete missing data, potentially resulting in a misinterpretation of the data. Instead of a constant, a central trend value for the feature, such as mean or median, may also be used. The most widely used strategy is to determine the missing value through approximate probabilistic models, using maximum likelihood procedures [60].

The presence of errors, or noise, in data is handled through two main approaches: data polishing and noise filters. Both results are similar, producing smooth output through regression techniques, outliers analysis, and data binning techniques. Extra significant computational overhead may be added depending on the complexity of the used model. Thus, it is necessary to strike a balance between the additional cost of cleaning the data and improving the accuracy of the obtained results [55].

The **data integration** techniques combine data from different sources, unifying the information and reducing redundancy. Redundancy is also eliminated through redundancy detection and data compression techniques, reducing the overhead of data transmission and storage. Redundancy may be spatial, temporal, statistical, or perceptual and is strongly present in video and image data. Techniques for handling redundancy in these cases are already well known, such as JPEG and PNG for image and MPEG-2, MPEG-4, H.263, and H.264/AVC for video. Nevertheless, they do not apply to the data analysis context over large-scale wireless networks. In this case, duplicate removal techniques are more appropriate.

[10]WizRule from WizSoft, available on https://www.wizsoft.com/products/wizrule/.
[11]IDEA products, available on https://idea.caseware.com/.
[12]DataSunrise Data Audit for MongoDB, available on https://www.datasunrise.com/audit/mongodb/
[13]Oracle Warehouse Builder 10g Release 2, available on https://www.oracle.com/technical-resources/articles/rittman-owb.html
[14]IBM SPSS, available on https://www.ibm.com/analytics/spss-statistics-software

During preprocessing, each instance can be composed of several features or attributes from different sources and which may have different schemas. Feature metadata becomes important to avoid errors during schema integration. Features that can be derived from other features or a feature set should be eliminated if they can be considered redundant. Redundancy is assessed by correlation analysis, which uses the *chi-square* test (χ^2) for nominal features, and the correlation and co-variance coefficients for numerical features. Data duplication must also be observed at the integration stage and data consistency must be ensured. Hence, during data integration it is necessary to detect and resolve conflicts in the feature values [60]. Two strategies for data integration include the use of Data Warehouse and Data Federation [55]. In the data warehouse, the integration happens through a three-stage process called Extraction, Transformation, and Loading (ETL) of data. In the *extraction*, the data required for the analysis is selected, then it is modified in the *transformation* stage by applying a set of rules, converting the data into a standard format. Finally, in the *loading* stage, the transformed data is imported into the storage infrastructure and usually comes from a single source [55]. In the data federation, a virtual database is created to query and aggregate data from different sources. The virtual database contains only information or metadata about the actual data and its location, not actually storing the data. These approaches are suitable for batch data processing but inefficient for streaming data.

Data reduction techniques aim to decrease the volume of the dataset while maintaining the result produced after processing. Techniques include strategies for reducing the number of dimensions and the number of samples. Data compression techniques are also used to reduce data volume. Dimension reduction seeks a compressed representation of the original data, focusing on decreasing the number of random variables or features considered. For this purpose, Wavelet Transforms, Principal Component Analysis (PCA), and Feature Selection (FS) are used. Genetic algorithms can also be employed to optimally reduce the size of the dataset [61]. In number reduction, the data is replaced by alternative, smaller-format representations, using parametric or non-parametric techniques. Parametrics, such as regression and log-linear models, estimate the data and store only the parameters of the model that describe it. Non-parametric techniques include histograms, clustering algorithms, sampling techniques, and data cube aggregation. Among the reduction techniques, it is worth discussing the wavelet transforms, the PCA method, and feature selection.

Wavelet Transform is a linear signal processing technique in which the signal is represented by a sum of simpler wave-forms. The wavelet transform aims to capture trends in numeric functions by decomposing the signal into a set of coefficients. Thus, the data vector is now represented by a vector of wavelet coefficients of the same length. There is a family of wavelet transforms that can be used for data preprocessing, the most popular being Haar-2, Daubechies-4, and Daubechcies-6 [62]. The lower coefficients may be discarded without significant impairment to the recovery of the original data. By storing a small fraction of the higher intensity coefficients, the original data is obtained by applying the inverse transform.

PCA creates a smaller set of variables that allows representing the original data and it is considered a feature extraction method. Data is described using a n-feature or n-dimension vector. PCA combines the essence of the original features using k orthogonal vectors of n dimensions, with $k \leq n$, which best represent the data to be reduced. As the dimensional projection space of the data is smaller, the number of dimensions of the data is reduced. In this new dimensional space, PCA reveals relationships that were previously unclear, allowing interpretations that would normally be ignored.

Feature Selection removes redundant or irrelevant features while maintaining a minimal set of features that results in a probability distribution close to the original distribution. Feature selection reduces the risk of overfitting data mining algorithms and reduces search space, making the learning process faster and consuming less memory [60]. Data is represented by a n-feature vector, which has 2^n possible subsets. From these subsets, the algorithm chooses an optimal amount to represent the data. Due to a large number of possibilities to be exploited to reach the optimal result, heuristics that exploit a small set of search spaces are generally used. In such cases, greedy algorithms are often used [63], which always choose local optimal solutions, such as step-wise forward selection, step-wise backward elimination, a combination of the previous two, and decision tree induction [64–66]. Another approach uses the chi-square independence test to decide which features should be selected.

Data transformation standardizes data representation and generally uses methods to reduce the complexity and dimensionality of the dataset [59], and may also be included in the data reduction step [60]. The main strategies for transforming data are smoothing techniques, feature construction, aggregation, normalization, discretization, and generation of concept hierarchies for nominal data. *Smoothing* removes noise from data using, for example, regression and grouping techniques. *Feature construction* aims at creating new features based on other features to improve the accuracy and understanding of large dimensional data. Along with feature selection, feature construction integrates a growing area of research known as feature engineering. Feature selection procedures identify and remove as much redundant and irrelevant information as possible [60], while extraction and construction combine the features of the original set to obtain a new set of less redundant variables. In *aggregation*, data are summarized resulting in a new representation. The summary relies on mean, variance, maximum, or minimum of values of the data. For example, daily energy consumption may be aggregated to show the maximum monthly or average annual consumption. *Normalization* changes the scale of the data to fit a smaller range of values. *Discretization* replaces the representation of numeric values with numeric ranges or conceptual labels. Finally, the *generation of concept hierarchies for nominal data* creates multiple levels of granularity into which data can be nested.

Although the area of data preprocessing is already well-explored, there is still a lot of research activity to seek for new methods and refine existing methods so that they can cope with the ever-increasing volume of data available and the need for real-time knowledge generation. Some works study new methods for, for example, splitting data between processors on cloud systems [67] and performing feature selection on streaming data [10].

4.4 Machine learning

After data preprocessing, big data analytics transforms data into knowledge using machine learning techniques. Machine learning applications are designed to identify and discover hidden patterns in data, to describe the result as a grouping of data in clustering problems, to predict the outcome of future events in classification and regression problems, and to evaluate the result of a data sample sequence for rule extraction problems

[53]. Thus, in a simplified way, there are four categories of machine learning problems: clustering, classification, regression, and rule extraction.

Clustering problems aim to minimize the distance between data samples with similar characteristics in the same group while maximizing the separation between distinct groups. *Classification* problems are characterized by mapping inputs into output target classes that are represented by the discrete set of output values. The cohesion of the results of the classifiers and clusters often uses as a parameter the Sum of Squared Errors (SSE), Eq. 1, which takes into account the number of clusters, k; the volume of data in the i-th cluster, n_i; the j-th data given in the i-th cluster, x_{ij}; the average of the data in the i-th cluster, c_i, as in Eq. 2; and the volume of data, n [59].

$$SSE = \sum_{i=1}^{k}\sum_{j=1}^{n_i} D(x_{ij} - c_i) \qquad (1) \qquad c_i = \frac{1}{n_i}\sum_{j=1}^{n_i} x_{ij}. \qquad (2)$$

The most common distance metric, D, is the Euclidean distance, defined in Eq. 3, where p_i and p_j are positions of two different data points. Other distances metrics are the Manhattan and Mikowski distances. Manhattan distance is suitable to work with high dimensional data, while Minkowski distance is used when the variables of interest are measured on ratio scales with an absolute zero value.

$$D(P - i, p_j) = \left(\sum_{l=1}^{d} |p_{il}, p_{jl}|^2\right)^{\frac{1}{2}} \qquad (3)$$

Metrics used to evaluate classification results are accuracy (Eq. 4), precision (Eq. 5), recall (Eq. 6), specificity 7, and F1 score (Eq. 8), which allow to find the hits, i.e., True Positives (TP) and True Negatives (TN), and the mismatches, i.e., False Positives (FP) and False Negatives (FN). Mismatches are known as FP when data that does not belong to the group is incorrectly classified as belonging to it, and as FN when data that belongs to the group is classified as not belonging to the group. This interpretation is summarized through a classifier's confusion matrix, as indicated in Table 3. Another important metric to evaluate classifiers is the Receiver Operating Characteristic (ROC) area, which is known as the Area Under the ROC Curve (AUC). The AUC checks the compromise between the true positive rate and the false positive rate. The AUC size is directly proportional to the classifier performance [56].

$$acc = \frac{TP + TN}{P + N}. \qquad (4)$$

$$p = \frac{TP}{TP + FP}. \qquad (5)$$

$$r = \frac{TP}{TP + FN}. \qquad (6)$$

$$e = \frac{TN}{FP + VN}. \qquad (7)$$

$$F = \frac{2pr}{p + r}. \qquad (8)$$

Regression problems tend to generate mathematical models, which, in turn, tend to behave as multivariate functions where input values are mapped to continuous output values. The *extraction* problems are different from the others, as the purpose of

Table 3 Confusion Matrix shows the interpretation of successful hits, and mismatches of a classifier

	Positive classification	**Negative classification**
Positive (P)	True positive (TP)	False Negative (FN)
Negative (N)	False Positive (FP)	True Negative (TN)

these problems is not to infer output values for each input set, but to identify statistical relationships between the data.

The main machine learning paradigms are supervised, unsupervised, semi-supervised, and reinforcement learning. Defining the learning paradigm to be used in a machine learning application determines how data should be collected, the establishment of the ground truth responsible for entering data labels, and the feature engineering responsible for establishing data features and what should be explored in the application. Figure 3 shows stream and batch processing with processes that represent the four machine learning paradigms. *Supervised learning* relies on a labeled dataset, called a training dataset, to create the data classification or regression model. This learning paradigm requires the existence, a priori, of a labeled dataset for model creation. In turn, the *semi-supervised learning* paradigm supports the use of training datasets with missing or incomplete labels. *Unsupervised learning*, on the other hand, seeks patterns in the training dataset and therefore does not require the existence of labeled data. Unsupervised learning is suitable for clustering problems where a data clustering pattern is sought to maximize the distance between groups while minimizing the distance between data within the same group. Finally, the *reinforcement learning* paradigm is an iterative process, in which agents effectively learn new knowledge in the absence of any explicit system model, with little or no knowledge of the environment [68]. The central idea of reinforcement learning is that agents learn based on examples from the training dataset that interact with the outside world. The learning process happens from reinforcements provided by the environment, which may be rewards or penalties. Thus, the training dataset consists of pairs of data samples and reinforcements, whether rewards or penalties. Feedback from the environment drives the agent to the best sequence of actions. Reinforcement learning is suitable for decision-making, planning, and programming problems [68].

Establishing the ground truth consists of the formal description, also called labels, given to the classes of interest. The methods for labeling datasets using the characteristics of a

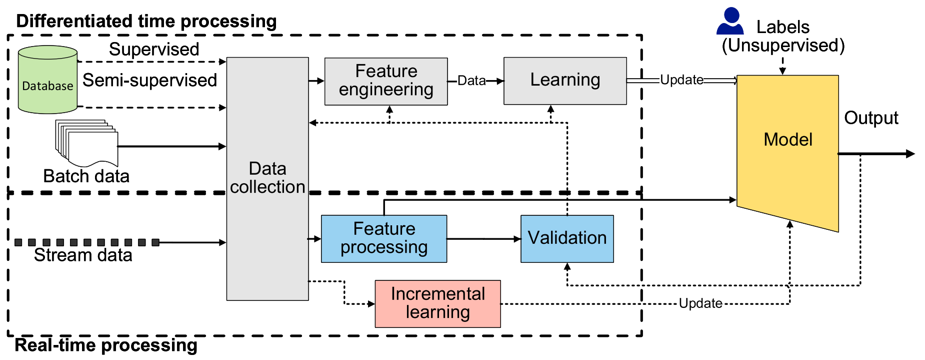


Fig. 3 Representation of the four machine learning paradigms in a batch and stream data processing system. The continuous and unlimited feature of data enables incremental learning where the model learns from incoming data. In batch processing, model training is only performed on stored historical databases

class are numerous. The most elementary method is manual labeling by specialists, with the help of specific tools that, for example, perform Deep Packet Inspection (DPI) [10], pattern matching as in intrusion detection systems, or unsupervised techniques such as raw data grouping [69]. An alternative to manual models is the development of statistical and structural models that describe data to infer a class of interest. However, the definition of ground truth in establishing the training set is closely related to the accuracy and precision of the machine learning model. The inherent mutual dependence of the training data size of one class of interest on the others impacts the performance of the model. Many machine learning techniques assume that interest classes have a similar distribution in the training dataset.

Machine learning is a powerful tool for gaining insight from data collected in a given scenario. Data from wireless networks can carry much hidden information. Specifically for Wi-Fi networks, by collecting and analyzing data, it is possible to discover network usage profiles [70, 71], determine the location of users [72], and monitor displacement [73], even allowing users to discover behavior patterns in user movement [74], activity recognition [75], and user identification [76].

5 Big data stream processing

The stream processing model expects data to arrive in real-time and, therefore, the system does not have control over order nor frequency in which sampĺes arrive. Samples arrives in a continuous and unbounded way, and the size and type of data are unknown. In real-time stream processing, the sample is processed only once since the system must be available for new data [56]. Samples can be temporarily stored in memory, but the memory is small compared to the potential size of the data arriving in the streams. Thus, to achieve the goal of real-time data processing, stream processing imposes restrictions on processing time per sample because, if the data arrival rate is greater than the processing service rate, the waiting queue for data to be processed grows indefinitely. As a consequence, data would be discarded. Therefore, one approach to provide efficient stream processing is to use distributed processing platforms and approximation techniques to speed up data processing.

Traditional machine learning methods are based on systems that assume that all collected data is fully loaded in a single batch and, thus, it can be centrally processed [77]. As the volume of data increases, however, the existing machine learning techniques fail when faced with unexpected volumes of data and also with the requirement to return the processing output as quickly as the data is generated. Thus there is a need for developing new methods of machine learning with faster response and adaptive behavior to meet the demands of processing big data in real-time [56].

In many cases, certain patterns and behaviors are lost or hidden in the middle of a large volume of data. Machine-learning-based systems help to discover this lost or hidden information. This is possible because, when new information becomes available, decision structures are reviewed and updated. Several models update their parameters considering one sample at a time. These models are: incremental learning, online learning, and sequential learning [78]. Incremental methods do not have time or sample order restrictions, while online methods require samples to be processed in order and only once, according to the time of arrival. Many incremental algorithms can be used in an online manner, but algorithms intend to model a behavior over time require samples to be in order.

In dynamic and non-stationary environments, the distribution of data may change over time, producing the phenomenon of concept drift. Concept drift refers to changes in the conditional distribution of the output, that is, the probability of belonging to a target class vary, given the vector of input features, while the distribution of the features remains unchanged [78]. An example of concept drift is a change on the customer consumption patterns. The buying preferences of a customer may change over time, depending on the day of the week, availability of products, or salary changes. In the security area, the models to detect threats become obsolete with minimal variations in the composition of the attacks [79]. The concept drift affects the performance of most learning algorithms making them less accurate over time. An effective predictor must be able to track these changes and quickly adapt to them. A hard problem when dealing with concept drift is to distinguish between noise and an actual change. Some algorithms excessively react to noise, misinterpreting it as concept drift. Others are highly robust to noise, adjusting to changes very slowly. The four different types of known concept drift are shown in Fig. 4: (i) sudden or abrupt change; (ii) incremental change; (iii) gradual change, and (iv) recurring or cyclical change.

Different approaches to detect concept drift can be used depending on the classification domain [10]. The first and simplest assumes that the data is static and, therefore, there is no change in the distribution of the data. It is possible to train the model only once and use the same model for future data. Another approach is to periodically update the static model with more recent historical data, also known as incremental learning. Some machine learning algorithms such as regression algorithms or neural networks make it possible to assess the importance of input data. In these algorithms, it is possible to use a weighting inversely proportional to the history of the data, so that more recent data is more important, with greater weight, and less recent data is less important, with smaller weight. Another approach is to use classifier sets algorithms such as AdaBoost or Random Forest. Thus, the static model remains intact, but a new model learns to correct the predictions of the static model based on the most recent data relationships. Finally, it is also possible to detect concept drift using heuristics or intrinsic data statistics. Heuristics such as accuracy or precision are mainly used in a supervised learning scenario in which data labels are present during training and classification. However, the presence of labels during classification is not usual in a production environment. In the unsupervised learning scenario, the statistical comparison of incoming samples, or the grouping of samples, with the samples used to train the system, assume that a concept drift is detected whenever new groups are found [80]. These methods of detecting changes tend to be more computationally intensive since measures based on distances are performed on the samples obtained.

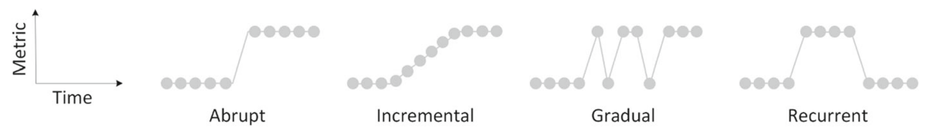


Fig. 4 Types of concept drift. In dynamic environments the data distribution can change over time, producing the phenomenon of concept drift)

5.1 Techniques for mining data streams

Gaber et al. categorize solutions for handling streams as data-based or task-based [81]. Data-based solutions aim to decrease the data representation, either through horizontal transformations, decreasing the number of features to handle, or vertical transformations, selecting a subgroup of samples to handle, also known as sampling. Task-based solutions focus on deploying computational techniques to find efficient solutions in terms of both time and storage space while data-based techniques rely on summarizing data or choosing subsets of data from the input data stream. Some of the data-based solutions are summarized as follows.

Data Sampling represents data samples as tuples, which are either selected for processing or discarded randomly. In the case of data arriving at a rate higher than the system can process promptly, sampling reduces the data arrival rate by discarding tuples. A possible usage scenario for data sampling is when collecting data on a high speed network. Instead of processing every single data sample, we process an approximate result of the collected data, without indefinitely increasing the queue of pending samples, and with less resource constraints than the complete operation [56]. The classic algorithm for maintaining online random sampling is the *reservoir sampling* technique. The algorithm maintains a sample of size s, called a reservoir. When new data streams arrive, each new element is likely to replace an old element in the reservoir. An extension of this algorithm is to keep samples of the most recent data of size k over a sliding window of size n.

Load Shedding refers to the process of discarding sequences of data streams when the input rate exceeds the processing capacity of the system. Thus, the system achieves an adaptive behavior to meet the latency requirements. This technique also causes loss of information. It generally applies to dynamic query systems. In data mining, load disposal is difficult to use, as the disposal of data blocks from the stream can lead to the loss of useful data for building models. It can also discard patterns of interest in a time series.

Sketching is the process of designing a random subset of attributes, or domain, of the data. The key idea is to produce faster results with mathematically proven error limits. The sketch does a vertical sampling, excluding attribute columns, of the data that arrives as a stream. Sketching's main disadvantage is the loss of accuracy because the results are an approximation. As an alternative to sketching on machine learning applications over data streams, there is the PCA, in which instead of using a subset of the attributes, a linear combination of attributes reduces the data dimensions while maximizing data variance.

Synopsis Structures are in-memory data summary structures. The key idea is to generate an approximated result while reducing memory complexity. The hash sketches proposal, for instance, creates a vector of bits with size L, where $L = log(N)$, and N is the number of data samples. Let $lsb(y)$ be the function that denotes the position of the least significant bit 1 in the binary representation of y. The incoming x data is mapped into a position of the bit vector using a uniform $hash(x)$ function and the $lsb(hash(x_0))$ function, which marks in the bit vector the occurrence of the data sample. From this proposal, it can be defined that since R is the position of the rightmost zero value in the bit vector, it is possible to estimate the number of elements in the bit vector as $E[R] = log(\phi d)$, where $\phi = 0.7735$, and $d = 2^R/\phi$. The use of histograms to estimate the relative frequency of samples in streams also constitute synopsis data structures.

Aggregation is the technique of summarizing incoming data. The summary may assume the form of average, variance, maximum, or minimum. The memory cost when using this technique is very low, but the technique fails if the data stream varies widely. It is worth mentioning as aggregation techniques the recursive average calculation, given by

$$\overline{x_i} = \frac{(i-1) \times \overline{x_{i-1}} + x_i}{i}, \tag{9}$$

where x_i is the current average, after i rounds, and the recursive variance calculation, given by

$$\sigma_i = \sqrt{\left(\sum x_i^2 - \left(\sum x_i\right)^2 / i\right) / (i-1)}. \tag{10}$$

Wavelets is a transform in which the signal is represented by a sum of waveforms, simpler and more defined in the construction of the transform, in different scales and positions. The wavelet transform aims to capture trends in numerical functions, decomposing the signal into a set of coefficients. Similar to the feature reduction using PCA, the lowest-order coefficients can be discarded. The reduced set of coefficients is used in algorithms that operate on the data stream. The most used transform in the context of data stream processing algorithms is *Haar wavelet*.

Task-based data processing techniques are methods that modify existing techniques, or create new ones, to meet the computational challenge of stream processing. The key task-based solutions follow.

Approximation algorithms return approximate computation results limited by error thresholds [81]. In streaming data mining, in particular, algorithms with approximate results are commonplace, as results are expected to be generated continuously, quickly, and with limited computational resources.

Time windows are commonly used to resolve queries on stream data with an undefined end. Instead of performing a computation on the complete data, it runs over a subset of data, possibly more than once over the same data. In this model, a timestamp is associated with each incoming data. The timestamp defines whether a data sample is inside or outside the window being considered. The computation is performed only on the data sample that is within the considered window. Some alternate approaches to time windows are the landmark window, the hopping window, the sliding window, and the tilted window.

The *landmark window* identifies relevant points in the data streams and, from there on, calculates the aggregation operators from that landmark. Successive windows share the same beginning and are of increasing size. A landmark may be, for instance, the start of a new day for daily data aggregation. The *hopping window* is a structure that considers a certain fixed number of samples and, when a subsequent set with a sufficient number of samples arrives, the previous ones are discarded, and computation is done on the new sample set. The hopping window uses the fixed sample size set only once, and each sample is only used in one set. The *sliding window*, in turn, is a data structure of fixed size that inserts more recent samples and removes the oldest samples in a similar way to the model of "first in, first out". Such a structure is computationally interesting since, in most cases, the whole past is not as relevant as the recent past. Thus, the sliding window considers only a fixed number of samples in the most recent past. The sliding window structure is widely used for calculating moving averages. Finally, the *tilted window* creates different resolutions for aggregating the data. Unlike previous windows, where samples were either inside the window or outside, in the skewed window, the most recent samples are treated with fine granularity, while samples from the past are grouped with lower granularity. The farther in the past, the coarser the grouping granularity of samples.

5.2 Online machine learning methods

A first approach for handling streaming big data is the use of learning methods capable of learning from infinite data in finite time. The central idea is to apply learning methods that limit the loss of information when using models with finite data in relation to models with infinite data. For this purpose, the loss of information is measured as a function of the number of samples used in each learning step and, then, the number of samples used in each step is minimized maintaining the loss threshold limit. The resolution of the problem of how much information is lost when decreasing the number of samples is given using the Hoeffding bound [78]. Considering a random variable x, whose value is contained in the interval R, it is assumed that independent observations of the variable are made and the mean $\bar{r}$ is computed. The Hoeffding bound ensures that, with probability $1 - \delta$, the true variable mean is given by at least $\bar{x} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{11}$$

The Hoeffding bound is independent of the distribution that generates the variable x. From this result, machine learning algorithms for training with stream data are developed. It is worth mentioning, however, that the values generated by the variable x are assumed to come from a stationary stochastic process. In cases where there is a change in the process that generates the variable used in the training of stream learning methods, a concept drift occurred and, therefore, it is necessary a new training of the learning method [82].

Typical approaches to learning new information involve maintaining the stochastic behavior of the data or discarding the existing classifier and, consequently, retraining with the data accumulated so far. Approaches that consider the end of the statistical stability of the data, i.e., the process is no longer stochastic, result in the loss of all previously acquired information. This phenomenon is known as catastrophic forgetfulness. Polikar et al. define that incremental learning algorithms must satisfy the following requirements: obtaining additional information from new data; not requiring access to the original data used to train the existing classifier; preserving previously acquired knowledge, i.e., it should not suffer catastrophic forgetting; and accommodating new classes that new data can introduce. Thus, classifiers that adopt incremental learning do not require training of the entire classifier in the event of a change in the steady behavior of the streaming data.

Incremental decision tree online algorithms

These algorithms are divided into two categories: (i) trees built using a greedy search algorithm, in which the addition of new information involves the complete restructuring of the decision tree, and (ii) incremental trees that maintain a sufficient set of statistics at each node of the tree to perform a node split test, making the classification more specific, when the accumulated statistics at the node are favorable to the division. An example of this type of incremental tree is the Very Fast Decision Tree (VFDT) algorithm [83]. The purpose of VFDT is to design a decision tree learning method for extremely large, potentially infinite datasets. The central idea is that each sample of information is read only once and in short processing time. It makes possible to directly manage online data sources, without storing samples. To find the best attribute that should be tested on a given node, it may be sufficient to consider only a small subset of the training samples that pass through that node. Thus, given a stream of samples, the first sample will be used to choose the root test; once the root attribute is chosen, subsequent samples are transmitted to the corresponding leaf nodes and used to choose the appropriate attributes on those nodes, and so on, recursively.

In a VFDT system, the decision tree is learned recursively, replacing leaves with decision nodes. Each leaf stores sufficient statistics on attribute values. Sufficient statistics are those required by a heuristic evaluation function that assesses the merit of node split tests based on attribute values. When a sample is available, it traverses the tree from the root to a leaf, evaluating the appropriate attribute on each node and following the branch corresponding to the attribute value in the sample. When the sample reaches the leaf, the statistics are updated. Then, the possible conditions based on the values of the attributes are evaluated. If there is sufficient statistical support in favor of a value test of one attribute concerning the others, the leaf is converted into a decision node. The new decision node will have as many descendants as possible values for the chosen decision attribute. Decision nodes maintain only information about the split test installed on the node. The initial state of the tree consists of a single leaf that is the root of the tree. The heuristic evaluation function is the Information Gain, denoted by $H(\cdot)$. The statistics that are sufficient to estimate the merit of a nominal attribute are the counters, n_{ijk}, that represent the number of examples of the class k that reach the leaf, where the attribute j receives the value i. The information gain measures the volume of information needed to classify a sample arriving at the node: $H(A_j) = inf(samples) - info(A_j)$. The j attribute information is given by

$$info(A_j) = \sum_i P_i \left(\sum_k -P_{ik} log_2(P_{ik}) \right), \tag{12}$$

where $P_{ik} = \frac{n_{ijk}}{\sum_a n_{ajk}}$ is the probability of observing the value of the attribute i given the class k, and $P_i = \frac{\sum_a n_{ija}}{\sum_a \sum_b n_{ajb}}$ is the probability of looking at the value of the i-th attribute.

In the VFDT system, the Hoeffding threshold given in Eq. 11 is used to decide how many samples are necessary to observe before installing a separation test on each leaf. Let $H(\cdot)$ be the attribute evaluation function for the information gain, then $H(\cdot)$ is $log_2(||K||)$, where K is the set of classes. Let x_a be the attribute with the highest value of $H(\cdot)$, x_b the attribute with the second-highest value of $H(\cdot)$, and $\overline{\Delta H} = \overline{H}(x_a) - \overline{H}(x_b)$, the difference between the two best attributes. Hence, if $\overline{\Delta H} > \epsilon$, with n samples observed on the leaf, the Hoeffding threshold defines with probability $1 - \delta$ that x_a is the attribute with the highest value in the evaluation function. Thus, the leaf must be transformed into a decision node that divides in x_a.

The function evaluation for each sample can be costly and, therefore, it is not efficient to compute $H(\cdot)$ on the arrival of each new sample. The VFDT proposal only computes the attribute evaluation function when a minimum number of samples, defined by the user, is observed since the last evaluation. When two or more attributes have the same values of $H(\cdot)$ continuously, even with a large number of samples, the Hoeffding threshold is not able to decide between them. Then, VFDT introduces a constant τ where $\overline{\Delta H} < \epsilon < \tau$. Hence, the leaf is converted into a decision node and the decision test is based on the best attribute. Gama et al. generalize the functioning of the VFDT system for numeric attributes [78].

Incremental Naive Bayes

Given a training set $\chi = (\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_n)$, where $\mathbf{x} \in \mathbb{R}^D$ are samples with D-dimensions in the attribute space and $y \in \{1, \cdots, K\}$ are the classes corresponding to a classification problem, Bayes' Theorem is formulated as

$$p(y = i|x) = \frac{p(i)p(\mathbf{x}|i)}{p(\mathbf{x})}, \tag{13}$$

where $p(i)$ is the a priori probability of a sample of the class occurring, and $p(y|\mathbf{x})$ is the unknown probability distribution of the $\mathbf{x}$ attribute space and marked with class i. An estimate for the unknown distribution is to assume the independence of the attributes given the marking of the class, leading to

$$p(x_1, x_2, \cdots, x_D|i)\ p(x_1|i)p(x_2|i) \cdots p(x_D|i), \tag{14}$$

where x_d represents the d-th dimension in the $\mathbf{x}$ attribute array. Thus, the Bayesian classifier is described as

$$F(\mathbf{x}) = \underset{i}{arg\ max} \prod_{d=1}^{D} p(x_d|i). \tag{15}$$

Hence, the classification is calculated by multiplying all the probabilities of the classes for each value of the sample attributes [84].

The incremental version of the Bayesian classifier predicts updating the values of the class probabilities by attributes according to the processing of new samples. One approach to allow efficient storage of probability functions as samples arrive is to discretize and store attribute histograms [10]. The Incremental Flexible Frequency Discretization (IFFD) proposal presents a method for discretizing quantitative attributes in a sequence of flexible size intervals [85]. This approach allows the insertion and division of intervals.

Incremental Learning by Classifier Aggregates

The Adaptive Ressonance Theory Mapping (ARTMAP) algorithm [86] is based on the generation of new decision clusters in response to new patterns that are sufficiently different from previously seen instances. The difference between an already known pattern and a new one is controlled by a user-defined surveillance parameter. Each grouping learns in a hyper-rectangle that is a different portion of the feature space, in an unsupervised way, which is then mapped to target classes. Clusters are always maintained as ARTMAP to avoid catastrophic forgetting. In addition, ARTMAP does not require access to previously seen data, and can accommodate new classes. However, ARTMAP is very sensitive to the selection of the surveillance parameter, the noise levels in the training data, and the order in which the training data arrives.

The AdaBoost algorithm generates a set of hypotheses and combines them through the voting of a weighted majority of the classes predicted by each individual hypotheses [87]. The hypotheses are generated by training a weak classifier[15] using instances extracted from a periodically updated distribution of training data. This distribution update ensures that instances poorly classified by the previous classifier are more likely to be included in

[15]Classification algorithms whose accuracy is close to random classification.

the training data of the next classifier. Thus, the training data of consecutive classifiers is aimed at instances that are increasingly difficult to classify.

The Learn++ incremental learning algorithm is inspired in AdaBoost and was originally developed to improve the classification performance of weak classifiers [88]. In essence, Learn++ generates a set of weak classifiers, each trained using a different distribution of training samples. The outputs of these classifiers are then combined using a majority voting regime to obtain the final classification rule. The use of weak classifiers is interesting, as the instabilities in building their decisions is sufficient for each decision to be different from the others so that small changes in their training datasets are reflected in different classifications.

Incremental K-Means

The k-means clustering algorithm performs an iterative optimization using the sum of the squared distances of all points at the center of each cluster. In the case of an unsupervised machine learning clustering problem, N is defined as the set of entries $\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_N$, where $\overline{x}_i \in R^n$. The problem then is to find $K << N$ clusters $c_1, c_2, \cdots, c_k$ with centers in $\overline{\omega}_1, \overline{\omega}_2, \cdots, \overline{\omega}_k$, respectively, such that the distance D from the squared sum of the data distances to the centers of the clusters to which they belong is minimal. The distance D is equivalent to the Mean Square Error (MSE) and is given by

$$D = \frac{1}{N} \sum_{j=1}^{K} \sum_{\overline{x} \in c_j} (\overline{x} - \overline{\omega}_j)^2. \quad (16)$$

The iterations of the classic algorithm consist of an initialization, in which K centers are chosen, and the elements are classified by the rule of the nearest-neighbor. Afterward, the centers of the clusters are updated by calculating the centroid of each cluster. In the following iterations, the data is reclassified according to the new calculated centroids. The iterations are performed until the convergence of the algorithm, which is achieved when the centers of the clusters calculated in iteration i are identical to those of iteration $i + 1$.

The adaptation of the k-means algorithm for sequential treatment of data consists of recalculating the centers of the clusters whenever a new sample arrives. This depends on all data already processed being accessed again to recalculate the centers of the clusters, which generates a substantially high demand for computational resources, making its use unfeasible. The variation of the algorithm using sequential blocks, in turn, enables the use of the k-means online algorithm since the clustering is performed on accumulated data blocks. Each block is used for l times of the k-means algorithm, and the results of the cluster centers of block i are used as the initial centers of the iteration over block $i + 1$. The incremental variation of the algorithm defines that the block is used only once [89]. The validity of the result of the incremental algorithm lies in the fact that the probability distribution of the sample attributes does not change, or changes slowly, between blocks.

5.3 Reinforcement learning

Reinforcement Learning (RL) is based on an agent interacting with the environment to learn an optimal policy by trial and error, for sequential decision-making problems in the fields of natural and social sciences and engineering [90]. The reinforcement learning agent interacts with the environment over time. At each step t of time, the agent applies a policy $\pi(a_t|s_t)$, where s_t is the state that the agent receives from a state space S, and a_t is an action selected by the agent in an action space A. The agent maps the state s_t into an action a_t to receive a reward, or penalty, r_t, escalate r_t, and transition to the next state s_{t+1}. The transition occurs according to the dynamics or the model of the environment. The function $R(s, a)$ models the agent's reward and the function $P(s_{t+1}|s_t, a_t)$ models the probability of transition between agent states. The agent continues the execution until it reaches a terminal state when the cycle is restarted. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the accumulated reward discounted by a factor $\gamma \in [0, 1)$. The central idea of reinforcement learning is that the agent maximizes the expectation of a long-term return for each state. It is worth mentioning that reinforcement learning mechanisms assume that the problem satisfies Markov's property, in which the future depends only on the current state and action, and not on the past. Thus, the problem is formulated as a Markov Decision Process (MDP), defined by the 5-tuple (S, A, P, R, γ).

Temporal difference learning is one of RL's pillars, as it refers to the learning methods for assessing the value function, StateActionRewardStateAction (SARSA) and Q-learning. Temporal difference learning discovers the value function $V(s)$ directly from experience using the time difference error, with model-free initialization, online, and fully incremental. Temporal difference learning is a prediction problem. The update rule is $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$, where α is the learning factor and $r + \gamma V(s') - V(s)$ is the so-called temporal difference error. The use of the generalized gradient descent algorithm guarantees the convergence of time difference learning problems [79].

The problem of policy forecasting or evaluation with reinforcement learning consists of calculating the state or action-value function for a policy. The control problem is to find the optimal policy. The SARSA algorithm evaluates the policy based on samples of the same policy and refines the policy using a greedy methodology with the action values. In off-policy methods, the agent learns a value function or ideal policy, maybe following an unrelated behavioral policy. The Q-learning algorithm, for example, tries to directly find action values for the optimal policy, not necessarily adjusting to the policy that generates the data. Thus, the policy found by Q-learning is generally different from the policy that generates the samples [90]. The notions of in-policy and off-policy methods relate to the ideas of evaluating the solutions found with the same policies that generate the data or evaluating with slightly different policies. Evaluating with different policies refers to using a model to generate the data, while in methods with no model, the agent learns through trial and error in the actions taken. Reinforcement learning algorithms can run in online or offline mode. In online mode, the training algorithms are executed on data acquired in sequential streams. In offline or batch mode, models are trained on the entire dataset.

Reinforcement learning is present in works that carry out knowledge extraction and decision making in wireless networks [91–94]. Liu and Yoo propose a scheme that uses the Q-learning algorithm to dynamically allocate blank subframes so that both LTE-U and Wi-Fi systems can successfully coexist [91]. The adjustment technique is based on reinforcement learning. The proposal introduces a new LTE-U frame structure, which in addition to allocating blank subframes, also reduces the LTE delay. In the proposal, the actions of the algorithm are modeled through a tuple that contains the total number of subframes in a block of frames and the portion of subframes for LTE-U. The states are modeled as the set of all actions taken up to that state. Tabrizi et al. argue that next-generation communications tend to use an integrated network system, in which Wi-Fi access points and cellular network base stations work together to maximize the QoS of the mobile user [92]. Thus, the authors propose that mobile devices with different access technologies easily switch from one access point or base station to another to improve user performance. The proposal is based on network selection to maximize QoS, formulated as a MDP. Therefore, they use an algorithm based on reinforcement learning designed to select the best network based on the current network load and also possible future network states. Chen and Qiu propose an approach for cooperative spectrum sensing using Software-Defined Radio (SDR) based on the Q-learning algorithm [93]. In sensing the spectrum using Q-learning, the state set consists of all combinations of a bit vector where each user sets a bit and the set of actions is $0, 1$, where "0" means that the channel is in the "available" state for secondary users, and an action with index "1" means that the channel is in the "busy" state, unavailable for secondary users. In the proposed algorithm, the rewards are positive when the action is in accordance with the occupation of the channel and negative, otherwise. Besides, Santos Filho et al. introduce a bandwidth control mechanism for cloud providers based on the Q-Learning algorithm [94]. Santos Filho et al. constantly adapt parameters on the cloud network infrastructure to meet the clients' Service Level Agreement, while the cloud provider maximizes the network occupancy and, thus, the provider's revenue.

5.4 Deep learning

Deep Learning (DL) is a class of machine learning techniques that explores multiple layers of nonlinear information processing to transform and extract higher-level information from original data [95]. It may be either supervised or unsupervised. In particular, DL is in the intersections among the research areas of neural networks, artificial intelligence, graphics modeling, optimization, pattern recognition, and signal processing. DL is generally used in the image, sound, and text processing [96].

The key idea of DL is to generate new representations of data for each layer, increasing the degree of abstraction of data representation. The increasing popularity of DL techniques occurs due to the accelerated increase in the processing capacity of chipsets, such as Graphic Processing Unit (GPU); the significant increase in the data available for training models; and recent advances in machine learning research [96], which has allowed the exploration of complex non-linear composition functions, distributed and hierarchical learning, and the effective use of labeled and non-labeled data.

DL architectures and techniques are used for data synthesis/generation or recognition/classification and, therefore, are generally classified in [97]: **Deep generating architectures**, which characterize the high order correlation properties of the observed data for pattern analysis or synthesis and/or joint statistical distributions characterization of the observed data and their associated classes. **Deep discriminative architectures**, which provide values to perform the discrimination of data into classes of patterns and, sometimes, characterizing the distributions *a posteriori* of classes conditioned to the observed data. **Deep hybrid architectures**, which discriminate data in classes assisted with the results of generating architectures through optimization and/or regularization, when discriminative criteria are used to learn the parameters in generative models. Generating architectures are associated with the identification and recognition of hidden patterns in observed data, while discriminative architectures are associated with the classification of observed data into defined classes. It is noteworthy that the generating architectures are related to problems of unsupervised learning, while discriminative architectures are related to problems of supervised learning. In the unsupervised learning process, there is no labeled data and, therefore, the main goal is to generate labeled data using unsupervised learning algorithms, such as Restricted Boltzmann Machine (RBM), Deep Belief Network (DBN), Deep Neural Network (DNN), generalized AutoEncoders, and Recurrent Neural Network (RNN) [96].

RBMs are generative probabilistic models capable of automatically extracting characteristics from the input data using completely unsupervised learning algorithm [98]. RBMs consist of a hidden layer and a visible layer of neurons with connections between the hidden and visible neurons represented by an array of weights. To train an RBM, samples from a training dataset are used as input to the RBM via visible neurons, and the network generates samples alternately back and forth between the visible and hidden neurons. The purpose of the training is to learn weights of the connections between visible and hidden neurons and to learn the neuron activation bias so that the RBM learns to reconstruct the input data during the phase when the visible neurons of the hidden neurons are sampled. Each sampling process is essentially a multiplication of matrices between a batch of training samples and the weight matrix, followed by a neuron activation function, which in many cases is $(1/(1 + e^{-x}))$. Sampling between the hidden and the visible layers is followed by a slight change in the parameters, defined by the learning rate α, and repeated for each batch of data in the training dataset, and for as many times as necessary to achieve convergence [98].

The **DBN** consists of multiple layers of stochastic and hidden variables and is related to the RBM, as it consists of the composition and stacking of several RBMs. The composition of multiple RBMs allows many hidden layers to efficiently train data through activations of an RBM for additional training stages [96, 98].

Convolutional Neural Network (CNN) represent one of the most common forms of **DNN** [99]. CNNs have multiple convolutional layers and each layer generates a map of characteristics, a higher-level abstraction of the input data, which preserves essential and unique information. Each of the convolutional layers in CNN presents, mostly, high-dimension convolutions, in which the input layer activations are structured as a set of input maps of characteristics, called a channel. For this reason, CNNs are generally used in signal processing. Each channel is converted with a different filter from the filter stack. The result of this computation is the output activations that creates an output characteristics map channel. Finally, several input feature maps can be processed together in batch to potentially improve the reuse of the filter weights.

AutoEncoder has traditionally been used for dimensionality reduction and feature learning. The fundamental idea of AutoEncoder is the presence of a hidden layer h that refers to the input, and two other main parts: the encoding function $h = f(x)$, and decoding or reconstruction function $r = g(h)$. The encoder and the decoder are trained together and the discrepancy between the original data and its reconstruction is then minimized. The deep AutoEncoder is a part of an unsupervised model [96].

Conventional Neural Networks are based on the principle that all data points are independent. For this reason, if data points are related in time or space, the chance of losing the state of the network is high. **RNN** are based on sequences so that they can model inputs or outputs composed of several independent elements. RNN can be used in unsupervised or supervised learning. When used in unsupervised learning, the prediction of the data sequence from previous data samples is possible, but it is difficult to train [96].

DL methods are usually trained with the descending stochastic gradient approach [79], in which a training example, with a known label, is used to update the model parameters. The strategy fits online stream learning. A strategy to speed up learning is to carry out updates on mini data batches instead of proceeding sequentially with one sample at a time [100]. The samples in each mini-batch are as independent as possible to provide a good balance between memory consumption and runtime.

Recent work use DL to infer the characteristics and behavior of wireless networks. Wang et al. developed an algorithm based on DL to explore bi-modal data [101]. The goal is to estimate the phase angle of arrival and average amplitudes on the 5 GHz radio interface for wireless networks. Their algorithm generates the indoor location fingerprints of devices. DL produces feature-based fingerprints from bi-modal data in the offline training stage. The weights in the deep AutoEncoder network are the fingerprints based on characteristics for each position. Wang et al. compare two indoor location approaches using DL, one with AutoEncoder and the other with Convolutional Neural Networks [102]. The authors conclude that the approach based on AutoEncoder presents less error in the inference of the indoor position. Turgut et al., in turn, use deep AutoEncoder to perform the indoor localization of devices, considering as initial characteristics the signal strength received from 26 access points [103]. Wang et al. use DL to recognize more accurate and robust activity on wireless channels. The central idea is to actively select available Wi-Fi channels with good quality and switch between adjacent channels to form an extended channel. Authors search for sequential patterns of channel usage, then, adopt a model of a Recursive Neural Network [104].

GPU improves performance in data consumption tasks using parallel computing. GPUs allow running a large number of threads in parallel, making it attractive for the computationally intensive tasks of state-space exploration such as DL [105]. OpenCL [106] and Compute Unified Device Architecture (CUDA)[107] are the main Application Programming Interface (API) to program and manage GPUs. CUDA platform, developed by Nvidia, and OpenCL give access to the GPU's virtual instruction set to run parallel tasks. The CUDA platform works with programming languages such as C, C ++, and Fortran. Some extensions are currently working with dynamically-typed languages, such as Python, with pycuda, pyopenCL [108], or numba [109]. A single GPU system computes a throughput of approximately 15 TFlops. Nevertheless, current solutions create a cluster of GPUs, increasing up to 2.6 times the speedup for a 4-GPU cluster [110], improving Distributed Deep Learning (DDL) tasks performance up to 5.5 times in a 100-GPU cluster[111].

Table 4 presents a qualitative comparison of the main techniques to process large streaming data discussed in this paper.

6 Big data processing tools

In wireless networks, stream processing is the most suitable model for big data processing due to its dynamic characteristics and the unrestricted generated data volume. In this scenario, new opportunities arise for industry and research to generate knowledge and intelligence. However, the large amount of data also poses a number of computational challenges, since the processing capacity of a single machine has not kept pace with the growth of data volume. Thus, to analyze wireless big data there is a need for a distributed

Table 4 Comparison between different knowledge extraction techniques. Stream processing applies different techniques to extract knowledge from arriving data. The techniques that best fit the envisioned mining scenario depends on the available data, on the training model, and on the expected goals of the processing

Technique	Data collection	Computing resource demand	Concept rift	Training	Supervised or unsupervised	Process	Typical applications	Related work
Mining	Sampling and Summarizing	Usually low	No	No	Unsupervised	Preprocessing	Knowledge extraction from historical databases	Gamaet al. [78], Gaber [81]
Online learning	Feature extraction	Medium	Yes	Yes	Supervised and Unsupervised	Classification, regression, or aggregation	Knowledge extraction from online transactions	Andreoni Lopez et al. [10], Lobato et al. [79], Hultenn et al. [83], Lu et al. [85], Masuyama et al. [86], Polikar [87], Polikar et al. [88], Aaron et al. [89]
Reinforcement learning	Model Free	Medium	Yes	Yes	Unsupervised	Reinforcement	Performance of controlling actions	Li et al. [90], Liu and Yoo [91], Tabrizi et al. [92], Chen and Qiu [93], Santos Filho et al. [94]
Deep Learning	Implicit feature extraction	High	Yes	Yes	Supervised and Unsupervised	Classification, regression, or aggregation	Image processing	Deng and Yu [95], Kwon et al. [96], Deng [97], Kim et al. [98], Sze et al. [99], Wang et al. [101], Wang et al. [102], Turgut et al. [103], Wang et al. [104]

processing system [112]. In this context, countless computational cluster programming models have emerged aiming at the treatment of several workloads [113–115].

Stream processing enables real-time data analysis, such as real-time log server file analysis or network flow analysis as packets reach devices. Batch data processing tools are used to work with static big data. The most widely used batch data processing tool is Hadoop [115]. Streaming data processing tools are required to analyze network data, as network flows that have a dynamic nature. Several open-source platforms for stream processing exist, such as Apache Spark Streaming [112], Apache Storm [116] and Apache Flink [117]. Apache Storm, and its upgraded version Apache Heron [118], are widely used, but Flink performs better and Spark has a superior failure recovery system [119, 120].

6.1 Apache spark

Apache Spark Streaming is a widely used distributed data processing tool for big data analytics. Spark is a batch distributed processing tool that has extensions to support a variety of workloads. Spark has Structured Query Language (SQL), machine learning, graph processing, and stream processing extensions using micro-batches concept [112]. Apache Spark also has YARN and Mesos modules for the computational cluster management, allowing the users to focus on new applications programming without worrying about infrastructure problems such as the location of data or on which node the data will be processed. Spark's general-purpose contributes to several benefits: (i) application development efficiency, provided by Spark's unified API; (ii) greater efficiency in processing tasks performance and memory storing. Spark performs many functions on the same data, while in regular memory, as opposed to other systems requiring nonvolatile memory data writing for access by other mechanisms; and (iii) possibility of creating new applications, such as interactive graphs queries and online machine learning, which is not possible with other tools.

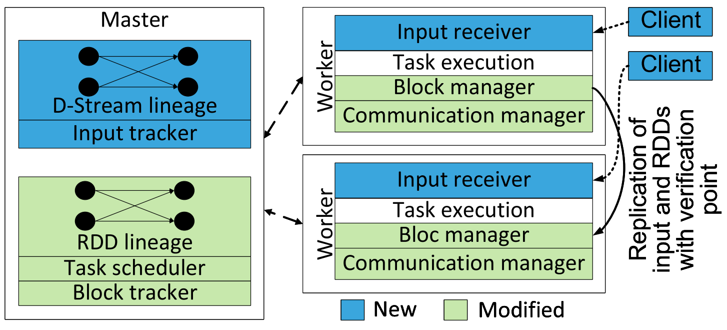
Spark has a programming paradigm similar to Hadoop's MapReduce but has its own data-sharing abstraction, called Resilient Distributed Dataset (RDD). RDD is Spark's core data structure and is responsible for memory data storage and distribution, ensuring resilience [56]. For data storage resilience, Spark can use HDFS or Simple Storage Service (S3), which are file systems for splitting, spreading, replicating, and managing data on computer clusters [56]. Thereby, stored disk data is distributed across multiple cluster nodes as well as data processed in memory, creating a fault-tolerant ecosystem. For each data transformation such as `map`, `filter`, and `groupBy`, a new RDD is created. Spark has an API for RDDs manipulation and is available in Scala, Java, Python, and R languages, where users execute local functions in the cluster. Data sharing is the main difference between Spark and other MapReduce-based platforms. Therefore, Spark is faster than other streaming platforms for interactive queries and iterative algorithms such as machine learning [121].

Distributed computing systems often provide fault tolerance through data replication or using checkpoints. Spark uses a different approach called lineage. Each RDD stores a graph containing all transformations used to build it and stores these operations in a database for future restoration in case of data loss. Lineage recovery is significantly more efficient than replication, saving time because transferring data across the network is slower than writing to RAM. Recovery is much faster than merely rerunning the program because the failed node often has multiple RDD partitions, and these partitions can be rebuilt in parallel with other nodes [122].

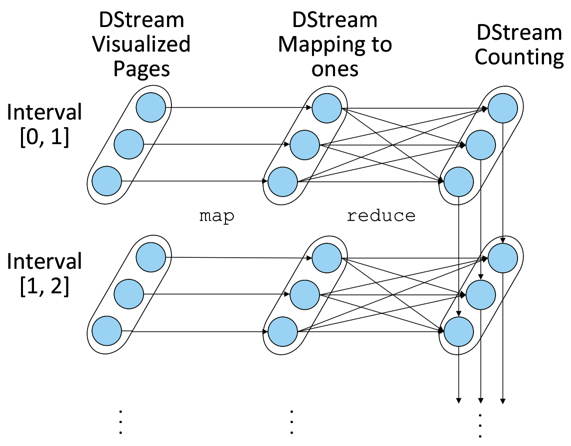
Apache Spark has four high-level libraries that use the RDD framework for a variety of functions. **SparkSQL** implements queries in SQL in a relational database. The central idea of this library is to have the same analytical database layout within the RDDs data structure. SparkSQL provides a high-level abstraction for data transformation called DataFrames, which are common abstractions for tabular data in Python and R, with programmatic methods for filters and aggregates. **GraphX** provides a graph-based operation computing interface similar to Pregel [123] and GraphLab [124] and implements the positioning optimization of these systems through its partitioning function option for RDD [125]. The **MLlib** is the library that implements machine learning functions and algorithms. It has classification, regression, clustering, collaborative filters, feature engineering, pipelines, among other functions. It allows machine learning algorithms to operate in a distributed fashion across a cluster, it is written in Scala and uses native C++ based on linear algebra libraries and has APIs for Java, Scala, and Python and is distributed as part of the Spark project [126]. It has a rich documentation describing all supported methods and sample code for each supported language.

The fourth high-level library enables Spark for streaming data processing. **Spark Streaming** implements a Discretized Streaming (D-Streams) processing model, which manages fault tolerance and stragglers data introduced by slower nodes [8]. Spark Streaming consists of three components: (i) The **Master** node, which monitors the D-Stream lineage graph and arranges the samples to calculate new RDD partitions; (ii) **Worker** nodes, which receive and process the data, and also store the input partitions and calculate the RDDs; and (iii) **Client** nodes, which send data to the Spark Master node. Spark Streaming modifies and adds several components to Spark to enable stream processing. Figure 5a shows the new and modified Spark Streaming components. Network communication, for instance, is rewritten to allow tasks with remote entries for fast recovery. There are modifications in the lineage mechanism, such as removing graphs from lineages that already have checkpoints, thus preventing graphs from growing indefinitely. From an architectural standpoint, the division of computational tasks into deterministic, short, or stateless instances is what sets Spark Streaming apart from other tools. Each task can be performed at any node in the cluster, or even at multiple nodes. D-Stream discretizes a data stream into small batches called micro-batches to the suitable computational tasks handle them.

Big data applications often use two approaches to tackle fault tolerance: replication and upstream backup. The *replication* approach uses two copies of the processed data and replicates the input records to other nodes. Nevertheless, replication alone is not enough. A synchronization protocol is also required to ensure each operator's copies perceive inherited data in the same order. Replication is computationally expensive, but recovery is fast. In the *upstream backup* approach, each node retains a data copy sent from a given checkpoint. When a node fails, a standby machine assumes its role, and the parent nodes reproduce messages to the previously standby machine to change its state. Both approaches have disadvantages. Replication has a large consumption of hardware resources, and upstream backup is slow to recover. Both approaches cannot handle straggler data. The D-Stream idea is to compute stateless stream data using deterministic batch computing at short time intervals. D-Stream computes the structures as follows: (i) the



(a) Spark Streaming architecture and its components. Adapted from [8].



(b) Lineage graph from a web page counting program. Adapted from [8].

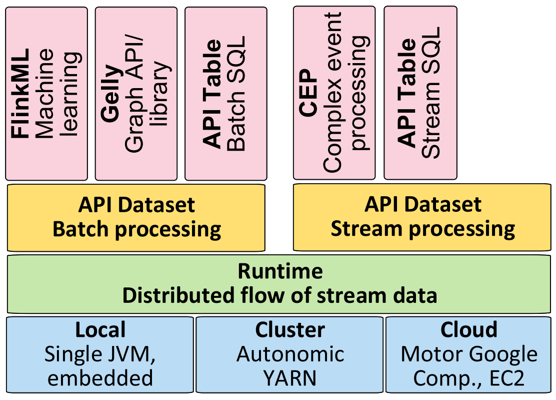
Fig. 5 Apache Spark Streaming architecture, and D-Stream lineage graph

state at each time step is deterministic to the input data, abolishing the need for synchronization protocols; (ii) the dependency between the current and the old state is granularly visible. D-Stream gives Spark Streaming an excellent recovery mechanism, similar to batch systems, that outperforms replication and upstream backup. D-Stream presents low latency because it uses the RDD data structure, which computes the data in memory and uses the lineage approach, as illustrated in Fig. 5b. D-Stream has rapid recovery from faults and straggler data because it uses determinism to provide *parallel recovery*. When a node fails, each cluster node works to compute again and retrieve the failed node's RDDs, resulting in significantly faster recovery than the other two traditional approaches [8]. D-Stream recovers from straggler data using speculative execution [115]. Each D-Stream is immutable, as RDDs, so each transformation generates a new D-Stream. A D-Stream is a set of RDDs sequences that can be influenced by the deterministic transformation.

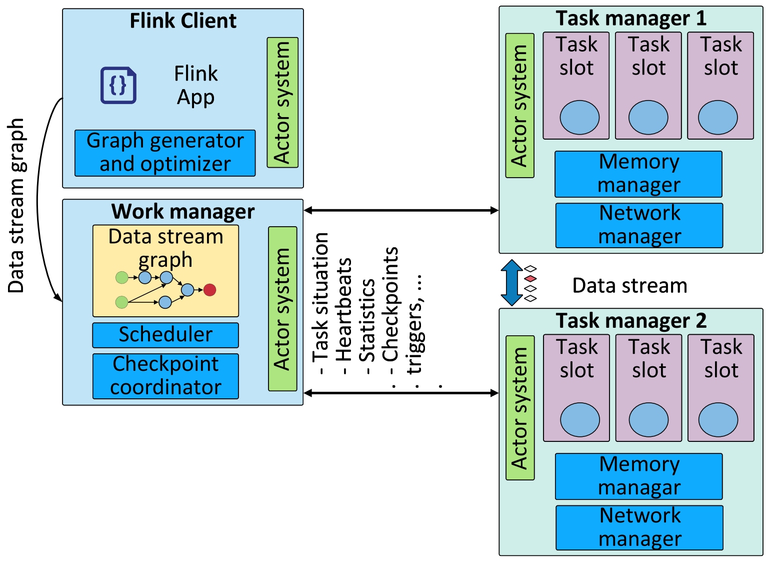
6.2 Apache flink

Apache Flink [117] supports stream and batch processing, making it a hybrid processing platform. Flink's architecture depicts four layers, named Development, Core, APIs, and Libraries, as shown in Fig. 6a.

Flink's core is the distributed data stream engine, which runs the developed code. Flink's analytics abstracts its tasks in a Directed Acyclic Graph (DAG) consisting of four components: sources, operators, sinks, and records that go through the graph. The graph in Fig. 7 consists of: (i) stateful operators and (ii) data streams, representing data produced by an operator and available for consumption by other operators. As data stream graphs run in parallel, operators are parallelized in one or more instances, called subtasks, and streams are divided into one or more partitions. State operators, which may be stateless in special cases, implement all processing logic, such as the operations: `filter`, `map`, `hash join`, `window`, etc. Streams distribute data between production and consumption operators in various patterns, such as peer-to-peer, broadcast, spread, fan-out, and merge.



(a) Flink architecture. Adapted from [117].



(b) MFlink Processing Model. Adapted from [117]

Fig. 6 Apache Flink architecture and working mode

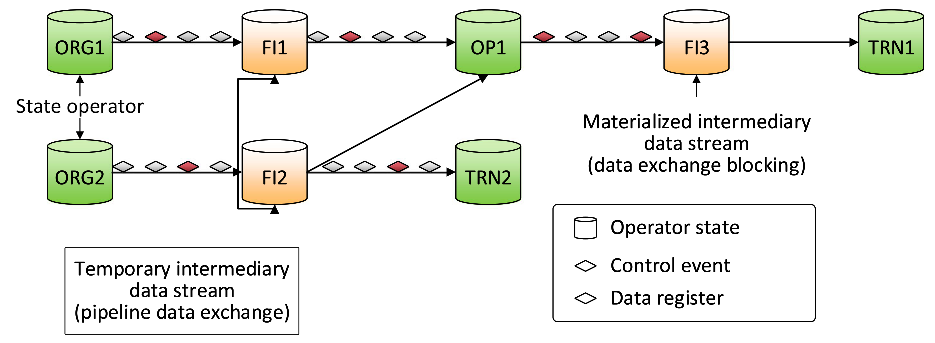

ORG1
FI1
OP1
FI3
TRN1
State operator
ORG2
FI2
TRN2
Materialized intermediary data stream (data exchange blocking)
Operator state
Control event
Data register
Temporary intermediary data stream (pipeline data exchange)

Fig. 7 Data flow graph. Adapted from [117]

Flink has two base APIs: the DataSet API for static and finite data processing, often associated with batch processing, and the DataStream API for dynamic and potentially unlimited data processing, often associated with stream processing. Flink has API specific libraries, FlinkML for machine learning, Gelly for graph processing, and Table for processing SQL operations.

A Flink cluster works on a master-slave fashion, as shown in Fig. **??**, and consists of three process types: Client, Job Manager, and at least one Task Manager. The *client* transforms the Flink program into a dataflow chart and sends it to the job manager. The *job manager* coordinates the data stream distributed execution. It tracks the status and progress of each operator and stream, schedules new operators, and coordinates checkpoints and recovery. In a high availability configuration, the job manager requires a minimal set of metadata at each checkpoint for fault-tolerant storage, so that a standby job manager can rebuild the checkpoint and resume execution from that point. Data processing occurs in *task managers*. A task manager runs one or more operators that produce streams and reports about their status to the job manager. Task managers maintain buffers for storing or creating streams and network connections for exchanging data streams between operators [117]. The data stream abstraction in Flink is called DataStream, and it consists of a partially ordered sequence of records. It is partially because there is no guarantee of order if an operator element receives more than one data stream as input [56].

Flink offers reliable execution ensuring the consistency of the "exactly-once" processing semantics, and it handles checkpoint failures and partial re-execution. The consistency of the "exactly-once" processing semantics is guaranteed through a verification mechanism that relies on distributed snapshots. The possibly limitless nature of a data stream turns unfeasible the recomputation on recovery, as likely it will take a long time to repeat a long task. To reduce the recovery time, Flink obtains a snapshot of operators, including the current position of the input streams at regular intervals. Flink uses a snapshot mechanism called Asyncrhonous Barrier Snapshotting (ABS) [117]. The barriers are control records injected into the input streams that correspond to a logical time, and logically split the stream into the part whose effects will be included in the current capture and the part that will be captured later. An operator receives upstream barriers and performs an alignment phase, making sure all input barriers have been received. Then, the operator writes its state, such as contents of a sliding window or a custom data structure in a durable file system, e.g., HDFS. Failure recovery reverts all operator states to their respective states taken from the last successful snapshot, and it restarts input streams from the last barrier for which there is a snapshot. The maximum load of recalculation required for recovery is limited to the number of entry records between two consecutive barriers. Moreover, partial recovery of a failed sub-task is possible by reproducing unprocessed records stored in buffers for upstream sub-tasks.

7 Discussion, trends, and research challenges

Despite the fast development of tools to deal with big data and streaming data, there are still many challenges to provide efficient wireless network real-time management and control. The main applications based on autonomous approaches for monitoring wireless network are anomaly detection; user, application, station, and network profiling; behavioral applications; network visualization; and optimization [45]. Network monitoring provides information about network elements and the network consolidated view. These data is used to detect problems and propose efficient solutions. Application monitoring provides information about network usage that enables resource planning and allocation. Station monitoring provides information about the user, network, and application usage patterns and it is used for network planning, access control, and for checking security policy violations. Application behavior classification allows the discrimination of the current applications, even when traffic is encrypted. Another application using the monitoring of streaming data is the inference of user identity based solely on the behavior of communication flows on the network. Indeed, the use of streaming data processing is an important component for providing network security by identifying anomalies or classifying traffic for worm signatures, port scanning, botnets, denial of service attacks, as well as the validation of new network policies.

There are still research challenges for handling big data in real-time. The main challenges of streaming data processing for wireless networks report to the five major dimensions of processing big data: volume, speed, variety, veracity, and value [100]. Each of these dimensions brings different issues when applied to the wireless network scenario. Other important challenges include:

- **Estimation of data errors**. It relates to the granularity of measurements made by network monitoring protocols such as SNMP, Netflow, and OpenFlow. Errors can occur in the sampling, transport, or collection processes and therefore are associated with data accuracy. The application of confidence interval estimation methods, good sampling practices, and error characterization have been studied and proposed to mitigate the errors in sampling network data [45].
- **Data significance**. It relates to the semantics of the data. Indeed, most monitoring data come from different sources and, therefore, the meaning of the collected data may be slightly different at each source. It can significantly affect the quality of machine learning results. Ontology, web semantic, and other techniques are proposed to mitigate such issues. Based on ontology modeling and semantic derivation, valuable patterns or rules can be discovered. Nevertheless, ontology and web semantic techniques are still not enough mature.
- **Pattern recognition training**. It consists of using labeled patterns to train learning algorithms. Obtaining labels, however, involves time and computational costs, particularly for streaming big data and carries the challenge of how to reduce the introduction of bias in the training. In addition, obtaining labels is impacted by both the volume and speed of the data stream. Another training-related challenge is the

balance between cost and accuracy, known as overfitting, which is still a critical open issue.

- **Data integration techniques**. They consist of the aggregation of data mining techniques, knowledge discovery, cloud computing, and machine learning. They relate to the variety of data, the speed with which new data is created, and the volume of streaming data arriving. A research trend is the adoption of hybrid approaches composed of several techniques, as each technique presents advantages and disadvantages.
- **Standard datasets and research environments**. They are essential for identifying current wireless network issues. Data collected from research environments can be used in supervised and unsupervised machine learning algorithms. Moreover, reinforcement learning can also benefit from artificially collected and labeled data [127].
- **Data security and privacy**. The use of data mining and machine learning technologies in big data processing to analyze personal information can produce overly relevant and interconnected results that undermine the privacy of individuals. Personal information not provided a priori can be inferred by correlating data from different sources. One of the key challenges of big data processing is to define efficient and effective methods that generate accurate knowledge while still ensuring the protection of sensitive personal information.

Other open challenges are identified by Sun et al. [127]. To implement supervised and unsupervised machine learning techniques in wireless networks, it is essential to create labeled/unlabeled datasets. For reinforcement learning-based approaches, network control problems should be constructed in well-defined environments. Transfer Learning promises to transfer knowledge learned from one scenario to another similar scenario, avoiding the need to train models from scratch for each scenario. Hence, the learning process in new environments is accelerated, and the machine learning algorithm can perform well even with a small amount of training data. Therefore, the transfer learning is fundamental for the practical implementation of learning models considering the cost of training without prior knowledge. Using transfer learning, network operators can solve new but similar problems without incurring high costs.

Heterogeneous backhaul/fronthaul machine-learning based controls can be applied in wireless network management. In future wireless networks, various backhaul/fronthaul solutions will co-exist, including fiber and cable, as well as wireless such as the sub-6 GHz band [127]. As each solution consumes different quantities of power and bandwidth, machine learning techniques can be used to select appropriate backhaul/fronthaul solutions based on the extracted traffic patterns and user performance requirements. In this sense, future updates of wireless network infrastructure will be developed based on optimization techniques supported by machine learning.

Another proposal to deal with the challenges of monitoring large-scale wireless networks is to slice the network on a per-service basis [19, 128, 129]. Hence, the network slices take different forms depending on each service in question. The main idea is to allocate appropriate resources, including computing resources, caching, backhaul/fronthaul, and radio on demand, to ensure the performance requirements of different isolated wireless network services, each one in an isolated slice. Network slicing benefits from machine learning in mapping service demands into resource allocation plans [94], and also employing transfer learning, as knowledge of resource allocation plans for different use cases in an environment can act as useful knowledge in another environment, accelerating the learning process.

7.1 Research projects

Research projects focus both on big stream data processing and on the development of large-scale experimentation networks based on wireless network technology. The experimentation infrastructures allow experimenting on federated networks including different types of wireless devices:

- **OneLab**[16] is a European initiative proposing to provide federated experimentation environments with different access technologies. OneLab brings together IoTLab-based experimentation environments, a wireless experimentation environment with Wi-Fi a/b/g/n nodes (NITOS), and the European PlanetLab Environment (PLE).
- **w-iLab.t**[17] is a wireless testbed developed under the Fed4FIRE testbed federation in Europe. It is located at IBBT office in Ghent, Belgium, and offers different wireless node technologies for experimentation, including SDR and mobile wireless nodes.
- **FIBRE**[18] is an open experimentation infrastructure that acts as a large-scale virtual lab for new applications and network architecture models. It is organized as a federation of 11 local experimentation islands, among which there are wireless network experimentation islands, such as the one available at the UFF, which also allows mobile indoor experimentation.
- **FIT**[19] is an open large-scale testing infrastructure for wireless and sensor systems and applications. FIT offers a wide range of technologies, such as the Internet of Things, wireless networking, and cloud computing, as well as a single system access interface and a large number of configuration and monitoring tools.
- **FUTEBOL**[20] is a cooperation project between Brazil and Europe to develop and implement an experimentation infrastructure that enables research on the convergence point between optical and wireless networks.
- **PrEstoCloud**[21] focuses on researching cloud computing technologies and real-time data analysis to provide a dynamic, distributed architecture for proactive data management. PrEstoCloud combines big data research, cloud computing, and real-time cloud computing.

Some projects provide frameworks to process big stream data. **Metron**[22] is a security analysis framework based on big data processing. The key idea of the framework is to allow the correlation of security events originating from different sources. To this end, the framework employs network sensors, action logs, and telemetry as data sources.

[16] Available at https://onelab.eu/.
[17] Available at https://www.fed4fire.eu/testbeds/w-ilab-t/
[18] Available at https://fibre.org.br/.
[19] Available at https://fit-equipex.fr/.
[20] Available at http://www.ict-futebol.org.br/.
[21] Available at http://prestocloud-project.eu/.
[22] Available at http://metron.apache.org/.

8 Conclusion

Wireless access networks are growing in number, variety, and speed. Several studies show that data from these networks are of critical value to network monitoring, management, and control. Nevertheless, these data also allows attackers to infer knowledge about users' infrastructure, mobility patterns, preferences, quality of experience, and services. The analysis of the data generated in a large-scale wireless access network is a problem of big data processing. The big data processing paradigm requires efficient storage, processing, and protection in terms of processing latency, memory storage space, and transmission bandwidth on the network. Moreover, large-scale wireless networks are sources of uninterrupted data that generate potentially infinite data with a wide variety of attributes. Therefore, it requires the use of streaming big data processing tools. In this sense, this article discussed key tools for wireless network monitoring and network analytics applications, such as machine learning, which is an important approach applied in classification, pattern discovery, and network security. We also discussed concepts related to streaming data processing and presented incremental machine learning algorithms. Incremental learning algorithms usually assume that the probability distribution of the attributes considered in the model does not vary over time. Nevertheless, in the event of a variation, i.e., a concept drift, either abrupt or dampened, incremental learning algorithms tend to fail to extract knowledge from incoming data. One of the key challenges for data stream processing, then, is the detection of changes in the statistics of attributes of the incoming data, and the development of machine learning algorithms that can respond to such changes.

In this article, we also analyzed the state-of-the-art applications of machine learning in wireless networks. Clusters of classifier, DL, and reinforcement learning are promising tools for knowledge extraction applications of future generation networks. There are new applications that adopt knowledge transfer learning, where knowledge extracted from one context is transferred to another application running in another context. Thus, knowledge transfer learning avoids training models from scratch and accelerates the learning process in new environments, because they perform well even with a small volume of data. Therefore, transfer learning is critical to the practical implementation of machine learning models, considering the cost of training without prior knowledge. Using transfer learning, network operators can solve new but similar problems more economically. Yet, the negative effects of prior knowledge on system performance requires further investigation. When considering DL proposals, the adoption of mixed techniques that combine deep and reinforcement learning is a trend. In addition, deep belief networks, which combine several layers of neural networks, are used in new applications, creating even deeper neural networks with more deep intermediate layers and different connections between neurons in each layer.

The stream processing of big data in wireless networks presents significant challenges for DL, due to large scale, heterogeneity, labeling noise, and non-stationary distribution of attributes. Thus, there is a need to address technical challenges with innovative proposals and transformative solutions to use the potential of streaming big data analytics in large-scale wireless networks. Research on the challenges posed by knowledge extraction in streaming big data is not only timely but necessary for many fields of knowledge beyond the management of wireless networks.

Acknowledgements
We acknowledge colleagues of the Universidade Federal Fluminense (UFF), Samsung R&D Institute, and Universidade Federal de Juíz de Fora (UFJF) for their incentives and suggestions.

Authors' contributions
All authors read and approved the final manuscript.

Funding
This research is partially funded by CNPq, CAPES, FAPERJ, and CGI/FAPESP.

Availability of data and materials
Please contact author for data requests.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
Theauthors declare that they have no competing interests.

Author details
[1]Universidade Federal Fluminense - UFF, Niterói, Rio de Janeiro Brazil. [2]Samsung R&D Institute Brazil - SRBR, Campinas, São Paulo Brazil. [3]Universidade Federal de Juiz de Fora - UFJF, Juiz de Fora, Minas Gerais Brazil.

Received: 16 March 2020 Accepted: 22 September 2020
Published online: 19 October 2020

References

1. Mattos D, Velloso P, Duarte O. An agile and effective network function virtualization infrastructure for the internet of things. J Internet Serv Appl. 2019;10(1):6. https://doi.org/10.1186/s13174-019-0106-y.
2. Cisco V. Global mobile data traffic forecast update, 2017–2022 white paper. Doc ID. 2019;1486680503328360:1–33.
3. Divgi G, Chlebus E. Characterization of user activity and traffic in a commercial nationwide wi-fi hotspot network: global and individual metrics. Wirel Netw. 2013;19(7):1783–805. https://doi.org/10.1007/s11276-013-0558-0.
4. Cici B, Gjoka M, Markopoulou A, Butts C. On the decomposition of cell phone activity patterns and their connection with urban ecology. In: Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing MobiHoc '15. New York: Association for Computing Machinery; 2015. p. 317–26.
5. Alessandrini A, Gioia C, Sermi F, Sofos I, Tarchi D, Vespe M. Wifi positioning and big data to monitor flows of people on a wide scale. In: 2017 European Navigation Conference (ENC). Lausanne: IEEE; 2017. p. 322–8.
6. Ishwarappa, Anuradha J. A brief introduction on big data 5vs characteristics and hadoop technology. Procedia Comput Sci. 2015;48:319–24. International Conference on Computer, Communication and Convergence (ICCC 2015).
7. Jang R, Cho D, Noh Y, Nyang D. Rflow+: An sdn-based wlan monitoring and management framework. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications; 2017. p. 1–9. https://doi.org/10.1109/INFOCOM.2017.8056995.
8. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I. Discretized streams: Fault-tolerant streaming computation at scale. In: XXIV ACM Symposium on Operating Systems Principles. New York: Association for Computing Machinery; 2013. p. 423–38. ACM.
9. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi† S, Tzoumas K. Apache flink: Stream and batch processing in a single engine. Bulletin IEEE Comput Soc Tech Comm Data Eng. 2015;4(34):28–38.
10. Andreoni Lopez M, Mattos D, Duarte O, Pujolle G. A fast unsupervised preprocessing method for network monitoring. Ann Telecommuns. 2019;74:. https://doi.org/10.1007/s12243-018-0663-2.
11. Guo J, Tan Z-H, Cho S, Zhang G. Wireless personal communications: Machine learning for big data processing in mobile internet. Wirel Pers Commun. 2018;102(3):2093–8. https://doi.org/10.1007/s11277-018-5916-x.
12. Acer U, Boran A, Forlivesi C, Liekens W, Pérez-cruz F, Kawsar F. Sensing wifi network for personal IoT analytics. In: 5th International Conference on the Internet of Things; 2015. p. 104–11. https://doi.org/10.1109/IOT.2015.7356554.
13. Acer U, Vanderhulst G, Masshadi A, Boran A, Forlivesi C, Scholl P, Kawsar F. Capturing personal and crowd behavior with wi-fi analytics. In: 3rd International Workshop on Physical Analytics. New York: Association for Computing Machinery; 2016. p. 43–8. ACM.
14. Gómez B, Coronado E, Villalón J, Riggio R, Garrido A. User association in software-defined wi-fi networks for enhanced resource allocation. In: 2020 IEEE Wireless Communications and Networking Conference (WCNC). Seoul: IEEE; 2020. p. 1–7.
15. Balbi H, Fernandes N, Souza F, Carrano R, Albuquerque C, Muchaluat-Saade D, Magalhaes L. Centralized channel allocation algorithm for IEEE 802.11 networks. In: 2012 Global Information Infrastructure and Networking Symposium (GIIS). Choroni: IEEE; 2012. p. 1–7.
16. Coronado E, Riggio R, Villalón J, Garrido A. Wi-balance: Channel-aware user association in software-defined wi-fi networks. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. Taipei: IEEE; 2018. p. 1–9.
17. Xu G, Gao S, Daneshmand M, Wang C, Liu Y. A survey for mobility big data analytics for geolocation prediction. IEEE Wirel Commun. 2017;24(1):111–9.

18. Chen Y, Guizani M, Zhang Y, Wang L, Crespi N, Lee G, Wu T. When traffic flow prediction and wireless big data analytics meet. IEEE Netw. 2018;161–167.
19. Isolani P, Cardona N, Donato C, Marquez-Barja J, Granville L, Latré S. SDN-based slice orchestration and MAC management for QoS delivery. In: 2019 Sixth International Conference on Software Defined Systems (SDS). Rome: IEEE; 2019. p. 260–5.
20. Luiz T, Freitas A, Guimarães F. A new perspective on channel allocation in wlan: Considering the total marginal utility of the connections for the users. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation GECCO '15. New York: ACM; 2015. p. 879–86. https://doi.org/10.1145/2739480.2754663.
21. Leung K, Kim B. Frequency assignment for IEEE 802.11 wireless networks. In: 2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484); 2003. p. 1422–63. https://doi.org/10.1109/VETECF.2003.1285259.
22. Maturi F, Gringoli F, Cigno R. A dynamic and autonomous channel selection strategy for interference avoidance in 802.11. In: 2017 13th Annual Conference on Wireless On-demand Network Systems and Services(WONS). Jackson: IEEE; 2017. p. 1–8.
23. Lin F, Wen Y, Fang L, Hsiao C. Resource allocation and multisession routing algorithms in coordinated multipoint wireless communication networks. IEEE Syst J. 2017;PP(99):1–12. https://doi.org/10.1109/JSYST.2017.2687102.
24. Shin M, Mishra A, Arbaugh W. Improving the latency of 802.11 hand-offs using neighbor graphs. In: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services MobiSys '04. New York: ACM; 2004. p. 70–83.
25. Zeljković E, Slamnik-Kriještorac N, Latré S, Marquez-Barja J. Abraham: Machine learning backed proactive handover algorithm using sdn. IEEE Trans Netw Serv Manag. 2019;16(4):1522–36.
26. Huang W, Chen Z, Dong W, Li H, Cao B, Cao J. Mobile internet big data platform in china unicom. Tsinghua Sci Technol. 2014;19(1):95–101.
27. Hadi M, Lawey A, El-Gorashi T, Elmirghani J. Big data analytics for wireless and wired network design: A survey. Comput Netw. 2018;132:180–99.
28. Wang T, Yang Q, Tan K, Zhang J, Liew S, Zhang S. Dcap: Improving the capacity of wifi networks with distributed cooperative access points. IEEE Trans Mob Comput. 2018;17(2):320–33. https://doi.org/10.1109/TMC.2017.2709743.
29. Ghouti L. Mobility prediction in mobile ad hoc networks using neural learning machines. Simul Model Pract Theory. 2016;66:104–21. https://doi.org/10.1016/j.simpat.2016.03.001.
30. Noulas A, Scellato S, Lathia N, Mascolo C. Mining user mobility features for next place prediction in location-based services. In: 12th International Conference on Data Mining. Brussels: IEEE; 2012. p. 1038–43.
31. Kulkarni V, Moro A, Garbinato B. Mobidict: A mobility prediction system leveraging realtime location data streams. In: Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming IWGS '16. New York: Association for Computing Machinery; 2016.
32. Stynes D, Brown K, Sreenan C. A probabilistic approach to user mobility prediction for wireless services. In: 2016 International Wireless Communications and Mobile Computing Conference (IWCMC). Paphos: IEEE; 2016. p. 120–5.
33. Zhang H, Dai L. Mobility prediction: A survey on state-of-the-art schemes and future applications. IEEE Access. 2019;7:802–22.
34. Bozkurt S, Elibol G, Gunal S, Yayan U. A comparative study on machine learning algorithms for indoor positioning. In: 2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA). Madrid: IEEE; 2015. p. 1–8.
35. Gonzalez M, Hidalgo C, Barabasi A-L. Understanding individual human mobility patterns. Nature. 2008;453(7196): 779.
36. Song C, Qu Z, Blumm N, Barabási A-L. Limits of predictability in human mobility. Science. 2010;327(5968):1018–21.
37. Toch E, Lerner B, Ben-Zion E, Ben-Gal I. Analyzing large-scale human mobility data: a survey of machine learning methods and applications. Knowl Inf Syst. 20181–23.
38. Qiao Y, Cheng Y, Yang J, Liu J, Kato N. A mobility analytical framework for big mobile data in densely populated area. IEEE Trans Veh Technol. 2017;66(2):1443–55.
39. Lin X, Li J, Baldemair R, Cheng J, Parkvall S, Larsson D, Koorapaty H, Frenne M, Falahati S, Grovlen A, Werner K. 5G new radio: Unveiling the essentials of the next generation wireless access technology. IEEE Commun Stand Mag. 2019;3(3):30–37.
40. Tariq F, Khandaker M, Wong K, Imran M, Bennis M, Debbah M. A speculative study on 6G. IEEE Wirel Commun. 2020;27(4):118–25.
41. Jiang C, Zhang H, Ren Y, Han Z, Chen K-C, Hanzo L. Machine learning paradigms for next-generation wireless networks. IEEE Wirel Commun. 2017;24(2):98–105.
42. Hofstede R, Čeleda P, Trammell B, Drago I, Sadre R, Sperotto A, Pras A. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. IEEE Commun Surv Tutor. 2014;16(4):2037–64.
43. Case J, Fedor M, Schoffstall M, Davin J. Simple Network Management Protocol (SNMP). RFC. 20111157.
44. Claise B, Sadasivan G, Valluri V, Djernaes M. Cisco systems netflow services export version 9. RFC. 20043954.
45. Li B, Springer J, Bebis G, Hadi Gunes M. Review: A survey of network flow applications. J Netw Comput Appl. 2013;36(2):567–81. https://doi.org/10.1016/j.jnca.2012.12.020.
46. Claise B, Trammell B, Aitken P. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC. 20137011.
47. Brownlee N. Flow-based measurement: Ipfix development and deployment. IEICE Trans Commun. 2011;94(8): 2190–8.
48. Medhi D, Ramasamy K. In: Medhi D, Ramasamy K, editors. Chapter 11 - routing and traffic engineering in software defined networks, Network Routing (Second Edition). Boston: Morgan Kaufmann; 2018, pp. 378–95. https://doi.org/10.1016/B978-0-12-800737-2.00013-2. http://www.sciencedirect.com/science/article/pii/B9780128007372000132.
49. Göransson P, Black C, Culver T. Chapter 5 - the openflow specification. In: Göransson P, Black C, Culver T, editors. Software Defined Networks (Second Edition). Second edition. Boston: Morgan Kaufmann; 2017. p. 89–136. https://doi.org/10.1016/B978-0-12-804555-8.00005-3. http://www.sciencedirect.com/science/article/pii/B9780128045558000053.

50. Krösche R, Thimmaraju K, Schiff L, Schmid S. I DPID-It My Way! A Covert Timing Channel in Software-Defined Networks. In: 2018 IFIP Networking Conference (IFIP Networking) and Workshops. Zurich: IEEE; 2018. p. 217–25.
51. Kim C, Sivaraman A, Katta N, Bas A, Dixit A, Wobker L. In-band network telemetry via programmable dataplanes. In: ACM SIGCOMM; 2015.
52. Kim Y, Suh D, Pack S. Selective in-band network telemetry for overhead reduction. In: 7th International Conference on Cloud Networking (CloudNet). Tokyo: IEEE; 2018. p. 1–3.
53. Boutaba R, Salahuddin M, Limam N, Ayoubi S, Shahriar N, Estrada-Solano F, Caicedo O. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. J Internet Serv Appl. 2018;9(1):16. https://doi.org/10.1186/s13174-018-0087-2.
54. Blei D, Smyth P. Science and data science. Proc Natl Acad Sci. 2017;114(33):8689–92. https://doi.org/10.1073/pnas.1702076114.
55. Hu H, Wen Y, Chua T, Li X. Toward scalable systems for big data analytics: A technology tutorial. IEEE Access. 2014;2:652–87.
56. Andreoni Lopez M, Mattos D, Duarte O, Pujolle G. Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. Concurr Comput Pract Experience. 2019;31(20):5344. https://doi.org/10.1002/cpe.5344.
57. Garofalakis M, Gehrke J, Rastogi R. Data Stream Management: Processing High-speed Data Streams, 1st edn. Berlin: Springer; 2016.
58. Lee G, Liu H, Yoon Y, Zhang Y. Improving sketch reconstruction accuracy using linear least squares method. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement IMC '05. Berkeley: USENIX Association; 2005. p. 24–24.
59. Tsai C-W, Lai C-F, Chao H-C, Vasilakos A. Big data analytics: a survey. J Big Data. 2015;2(1):21.
60. García S, Ramírez-Gallego S, Luengo J, Benítez J, Herrera F. Big data preprocessing: methods and prospects. Big Data Analytics. 2016;1(1):9.
61. Tannahill B, Jamshidi M. System of systems and big data analytics–bridging the gap. Comput Electr Eng. 2014;40(1):2–15.
62. Han J, Pei J, Kamber M. Data Mining: Concepts and Techniques. Massachusetts: Elsevier; 2011.
63. Vafaie H, Imam I. Feature selection methods: genetic algorithms vs. greedy-like search. In: Proceedings of the International Conference on Fuzzy and Intelligent Control Systems, vol. 51. Louisville; 1994. p. 28.
64. Aminanto M, Choi R, Tanuwidjaja H, Yoo P, Kim K. Deep abstraction and weighted feature selection for Wi-Fi impersonation detection. IEEE Trans Inf Forensics Secur. 2018;13(3):621–36.
65. Tsamardinos I, Borboudakis G, Katsogridakis P, Pratikakis P, Christophides V. A greedy feature selection algorithm for big data of high dimensionality. Mach Learn. 2019;108(2):149–202.
66. Wang C, Huang Y, Shao M, Hu Q, Chen D. Feature selection based on neighborhood self-information. IEEE Trans Cybernet. 2020;50(9):4031–42.
67. Ham Y, Lee H-W. Big data preprocessing mechanism for analytics of mobile web log. Int J Adv Soft Comput Appl. 2014;6(1).
68. Tesauro G. Reinforcement learning in autonomic computing: A manifesto and case studies. IEEE Internet Comput. 2007;11(1):22–30. https://doi.org/10.1109/MIC.2007.21.
69. Zhang X, Wang C, Li Z, Zhu J, Shi W, Wang Q. Exploring the sequential usage patterns of mobile internet services based on markov models. Electron Commer Res Appl. 2016;17:1–11. https://doi.org/10.1016/j.elerap.2016.02.002.
70. Afanasyev M, Chen T, Voelker G, Snoeren A. Usage patterns in an urban wifi network. IEEE/ACM Trans Netw (ToN). 2010;18(5):1359–72.
71. Fan Y, Chen Y, Tung K, Wu K, Chen A. A framework for enabling user preference profiling through wi-fi logs. IEEE Trans Knowl Data Eng. 2016;28(3):592–603.
72. Huang H, Cai Y, Yu H. Distributed-neuron-network based machine learning on smart-gateway network towards real-time indoor data analytics. In: Proceedings of the 2016 Conference on Design, Automation & Test in Europe DATE '16. Dresden: IEEE; 2016. p. 720–5.
73. Chilipirea C, Petre A, Dobre C, van Steen M. Presumably simple: Monitoring crowds using wifi. In: Proceedings of the 17th IEEE International Conference on Mobile Data Management (MDM), vol. 1. Porto: IEEE; 2016. p. 220–5.
74. Zeng Y, Pathak P, Mohapatra P. Analyzing shopper's behavior through wifi signals. In: Proceedings of the 2Nd Workshop on Physical Analytics WPA '15. New York: Association for Computing Machinery; 2015. p. 13–8.
75. Alsheikh M, Niyato D, Lin S, Tan H-P, Han Z. Mobile big data analytics using deep learning and apache spark. IEEE Netw. 2016;30(3):22–9.
76. Zeng Y, Pathak P, Mohapatra P. WiWho: WiFi-Based Person Identification in Smart Spaces. In: Proceedings of the 15th International Conference on Information Processing in Sensor Networks IPSN '16. Vienna: IEEE; 2016. p. 4–1412.
77. Qiu J, Wu Q, Ding G, Xu Y, Feng S. A survey of machine learning for big data processing. EURASIP J Adv Sig Process. 2016;2016(1):67. https://doi.org/10.1186/s13634-016-0355-x.
78. Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A. A survey on concept drift adaptation. ACM Comput Surv (CSUR). 2014;46(4):44.
79. Lobato A, Andreoni Lopez M, Sanz I, Cardenas A, Duarte O, Pujolle G. An adaptive real-time architecture for zero-day threat detection. In: 2018 IEEE International Conference on Communications (ICC). Kansas City: IEEE; 2018. p. 1–6.
80. Jordaney R, Sharad K, Dash S, Wang Z, Papini D, Nouretdinov I, Cavallaro L. Transcend: Detecting concept drift in malware classification models. In: PROCEEDINGS OF THE 26TH USENIX SECURITY SYMPOSIUM. Vancouver: USENIX Association; 2017. p. 625–42.
81. Gaber M. Advances in data stream mining. Wiley Interdiscip Rev Data Min Knowl Disc. 2012;2(1):79–85.
82. Wang S, Minku L, Ghezzi D, Caltabiano D, Tino P, Yao X. Concept drift detection for online class imbalance learning. In: The 2013 Int. Joint Conference on Neural Networks (IJCNN). Dallas: IEEE; 2013. p. 1–10. https://doi.org/10.1109/IJCNN.2013.6706768.
83. Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '01. New York: Association for Computing Machinery; 2001. p. 97–106. https://doi.org/10.1145/502512.502529.

84. Godec M, Leistner C, Saffari A, Bischof H. On-line random naive bayes for tracking. In: 2010 20th Int. Conference on Pattern Recognition; 2010. p. 3545–8. https://doi.org/10.1109/ICPR.2010.865.
85. Lu J, Yang Y, Webb G. Incremental discretization for naïve-bayes classifier. In: Li X, Zaïane O, Li Z, editors. Advanced Data Mining and Applications. Berlin: Springer; 2006. p. 223–38.
86. Masuyama N, Loo C, Dawood F. Kernel bayesian art and artmap. Neural Netw. 2018;98:76–86.
87. Polikar R. Ensemble based systems in decision making. IEEE Circ Syst Mag. 2006;6(3):21–45.
88. Polikar R, Upda L, Upda S, Honavar V. Learn++: an incremental learning algorithm for supervised neural networks. IEEE Trans Syst Man Cybern Part C Appl Rev. 2001;31(4):497–508. https://doi.org/10.1109/5326.983933.
89. Aaron B, Tamir D, Rishe N, Kandel A. Dynamic incremental k-means clustering. In: 2014 International Conference on Computational Science and Computational Intelligence; 2014. p. 308–13. https://doi.org/10.1109/CSCI.2014.60.
90. Li Y. Deep Reinforcement Learning. arXiv e-prints. 20181810–06339. 1810.06339.
91. Liu Y, Yoo S. Dynamic resource allocation using reinforcement learning for LTE-U and WiFi in the unlicensed spectrum. In: 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN). Milan: IEEE; 2017. p. 471–5.
92. Tabrizi H, Farhadi G, Cioffi J. A learning-based network selection method in heterogeneous wireless systems. In: 2011 IEEE Global Telecommunications Conference - GLOBECOM 2011; 2011. p. 1–5.
93. Chen Z, Qiu R. Cooperative spectrum sensing using q-learning with experimental validation. In: 2011 Proceedings of IEEE Southeastcon; 2011. p. 405–8.
94. Santos Filho R, Ferreira T, Mattos D, Medeiros D. A lightweight reinforcement-learning-based mechanism for bandwidth provisioning on multitenant data center. In: Proceedings of the International Conference on Systems, Signals and Image Processing (IWSSIP). Niterói: IEEE; 2020. p. 1–8.
95. Deng L, Yu D. Deep learning: Methods and applications. Found Trends Sig Process. 2014;7(3-4):197–387. https://doi.org/10.1561/2000000039.
96. Kwon D, Kim H, Kim J, Suh S, Kim I, Kim K. A survey of deep learning-based network anomaly detection. Clust Comput. 2017;161–167.
97. Deng L. A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA Trans Sig Inf Process. 2014;3:2. https://doi.org/10.1017/atsip.2013.9.
98. Kim S, McMahon P, Olukotun K. A large-scale architecture for restricted boltzmann machines. In: 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines; 2010. p. 201–8. https://doi.org/10.1109/FCCM.2010.38.
99. Sze V, Chen Y, Yang T, Emer J. Efficient processing of deep neural networks: A tutorial and survey. Proc IEEE. 2017;105(12):2295–329.
100. Chen X, Lin X. Big data deep learning: Challenges and perspectives. IEEE Access. 2014;2:514–25.
101. Wang X, Gao L, Mao S. BiLoc: Bi-Modal Deep Learning for Indoor Localization With Commodity 5GHz WiFi. IEEE Access. 2017;5:4209–20. https://doi.org/10.1109/ACCESS.2017.2688362.
102. Wang X, Wang X, Mao S. RF sensing in the internet of things: A general deep learning framework. IEEE Commun Mag. 2018;56(9):62–7. https://doi.org/10.1109/MCOM.2018.1701277.
103. Turgut Z, Üstebay S, Zeynep Gürkaş Aydın G, Sertbaş A. In: Boyaci A, Ekti A, Aydin M, Yarkan S, editors. Deep learning in indoor localization using WiFi. Singapore: Springer; 2019, pp. 101–10.
104. Wang F, Gong W, Liu J, Wu K. Channel selective activity recognition with WiFi: A deep learning approach exploring wideband information. IEEE Trans Netw Sci Eng. 20181–1. https://doi.org/10.1109/TNSE.2018.2825144.
105. Huynh L, Lee Y, Balan R. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. In: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services. New York: Association for Computing Machinery; 2017. p. 82–95.
106. Stone J, Gohara D, Shi G. Opencl: A parallel programming standard for heterogeneous computing systems. Comput Sci Eng. 2010;12(3):66–73.
107. NVIDIA Corporation. NVIDIA CUDA C Programming Guide. 2010;3.1.1.
108. Klöckner A, Pinto N, Lee Y, Catanzaro B, Ivanov P, Fasih A. Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation. Parallel Comput. 2012;38(3):157–74.
109. Lam S, Pitrou A, Seibert S. Numba: a LLVM-based Python JIT compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. New York: Association for Computing Machinery; 2015. p. 1–6.
110. Sun Y, Baruah T, Mojumder S, Dong S, Gong X, Treadway S, Bao Y, Hance S, McCardwell C, Zhao V, et al. MGPUSim: enabling multi-GPU performance modeling and optimization. In: Proceedings of the 46th International Symposium on Computer Architecture. New York: Association for Computing Machinery; 2019. p. 197–209.
111. Gu J, Chowdhury M, Shin K, Zhu Y, Jeon M, Qian J, Liu H, Guo C. Tiresias: A {GPU} cluster manager for distributed deep learning. In: 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19); 2019. p. 485–500.
112. Zaharia M, Das T, Li H, Shenker S, Stoica I. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In: Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing. Boston: USENIX Association; 2012. p. 10–10.
113. Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein J. Distributed graphlab: a framework for machine learning and data mining in the cloud. Proc VLDB Endowment. 2012;5(8):716–27.
114. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Oper Syst Rev. 2007;41(3):59–72. ACM.
115. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
116. Iqbal M, Soomro T. Big data analysis: Apache storm perspective. Int J Comput Trends Technol. 2015;19(1):9–14.
117. Carbone P, Fóra G, Ewen S, Haridi S, Tzoumas K. Lightweight asynchronous snapshots for distributed dataflows. arXiv preprint arXiv:1506.08603. 2015;1–8.
118. Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel J, Ramasamy K, Taneja S. Twitter heron: Stream processing at scale. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. New York: Association for Computing Machinery; 2015. p. 239–50.

119. Andreoni Lopez M, Lobato A, Duarte O. A performance comparison of open-source stream processing platforms. In: 2016 IEEE Global Communications Conference (GLOBECOM). Washington: IEEE; 2016. p. 1–6.
120. Chintapalli S, Dagit D, Evans B, Farivar R, Graves T, Holderbaugh M, Liu Z, Nusbaum K, Patil K, Peng B, et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Chicago: IEEE; 2016. p. 1789–92.
121. Xin R, Rosen J, Zaharia M, Franklin M, Shenker S, Stoica I. Shark: SQL and rich analytics at scale. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: Association for Computing Machinery; 2013. p. 13–24.
122. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin M, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. San Jose: USENIX Association; 2012. p. 2–2.
123. Malewicz G, Austern M, Bik A, Dehnert J, Horn I, Leiser N, Czajkowski G. Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. New York: Association for Computing Machinery; 2010. p. 135–46.
124. Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein J. Graphlab: A new framework for parallel machine learning. arXiv preprint arXiv:1408.2041. 2014.
125. Gonzalez J, Xin R, Dave A, Crankshaw D, Franklin M, Stoica I. GraphX: Graph processing in a distributed dataflow framework. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). Broomfield: USENIX Association; 2014.
p. 599–613.
126. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, et al. Mllib: Machine learning in apache spark. J Mach Learn Res. 2016;17(1):1235–41.
127. Sun Y, Peng M, Zhou Y, Huang Y, Mao S. Application of machine learning in wireless networks: Key techniques and open issues. Technical report. 2018arXiv preprint arXiv:1809.08707. https://arxiv.org/abs/1809.08707. Acessado em 22 Mar 2019.
128. Foukas X, Patounas G, Elmokashfi A, Marina M. Network slicing in 5g: Survey and challenges. IEEE Commun Mag. 2017;55(5):94–100.
129. Sanz I, Mattos D, Duarte O. SFCPerf: An automatic performance evaluation framework for service function chaining. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. Taipei: IEEE; 2018. p. 1–9.

Publisher's Note