On the Classification of Fog computing applications: A Machine Learning Perspective

Judy C. Guevara[1], Ricardo da S. Torres[2], Nelson L. S. da Fonseca[1]*

[1]*Institute of Computing - University of Campinas, Campinas 13083-852, SP, Brazil*
[2]*Department of ICT and Natural Sciences, Norwegian University of Science and Technology (NTNU), Ålesund, Norway*

Abstract

Currently, Internet applications running on mobile devices generate a massive amount of data that can be transmitted to a Cloud for processing. However, one fundamental limitation of a Cloud is the connectivity with end devices. Fog Computing overcomes this limitation and supports the requirements of time-sensitive applications by distributing computation, communication, and storage services along the Cloud to Things (C2T) continuum, empowering potential new applications, such as smart cities, augmented reality (AR), and virtual reality (VR). However, the adoption of Fog-based computational resources and their integration with the Cloud introduces new challenges in resource management, which requires the implementation of new strategies to guarantee compliance with the quality of service (QoS) requirements of applications.

In this context, one major question is how to map the QoS requirements of applications on Fog and Cloud resources. One possible approach is to discriminate the applications arriving at the Fog into Classes of Service (CoS). This paper thus introduces a set of CoS for Fog applications which includes, the QoS requirements that best characterize these Fog applications. Moreover, this paper proposes the implementation of a typical machine learning classification methodology to discriminate Fog Computing applications as a function of their QoS requirements. Furthermore, the application of this

*Corresponding author

Email addresses: `jguevara@lrc.ic.unicamp.br` (Judy C. Guevara[1]), `ricardo.torres@ntnu.no` (Ricardo da S. Torres[2]), `nfonseca@ic.unicamp.br` (Nelson L. S. da Fonseca[1])

methodology is illustrated in the assessment of classifiers in terms of efficiency, accuracy, and robustness to noise. The adoption of a methodology for machine learning-based classification constitutes a first step towards the definition of QoS provisioning mechanisms in Fog computing. Moreover, classifying Fog computing applications can facilitate the decision-making process for Fog scheduler.

Keywords: Fog Computing, Edge Computing, Cloud Computing, Internet of Things, Scheduling, Classes of Service, Quality of Service, Machine Learning, Feature Selection, Attribute noise, Classification Algorithms.

1. Introduction

Cloud Computing enables ubiquitous access to shared pools of configurable resources and services over the Internet that can be rapidly provisioned with minimal management effort [1]. However, with the increasing relevance of the Internet of Things (IoT), mobile and multimedia applications, the transfer delays between the Cloud and an end device have been deemed too long and not suitable for latency-sensitive applications, making the main limitation in the use of the Cloud [2, 3] for latency-sensitive and mobile applications [4, 5, 6].

The plethora of applications running on the Internet has heterogeneous processing and communications demands, as well as Quality of Service (QoS) requirements. While multimedia demand processing power and storage space [7], others such as mission-critical require strict response time. Mobile users need to have continuous access to applications when on the move. Moreover, IoT devices and sensors generate large amounts of data. Not all data need to be sent to the Cloud while some data have to be processed immediately.

Fog computing aims at coping with these demands by hosting Cloud services on connected heterogeneous devices, typically, but not exclusively located at the edge of the network [8, 9]. The Fog provides a geographically distributed architecture for computation, communication, and storage, which targets real-time applications and mobile services. End-users benefit from pre-processing of workloads, geo-distribution of resources, low latency responses, device heterogeneity [8], and location/content awareness [10]. The Fog can support the diversity of applications requirements in the Cloud to Things (C2T) continuum, which is comprised of end devices, one or more levels of Fog nodes, and the Cloud. Fog nodes located at the edge of the network are usually limited in resources [11]. Still, their use involves only brief delays in communication while the Cloud has a large ("unlimited") number of resources, but involves long delays in communication. On the lowest level of this continuum, the initial processing can be carried out, and results passed on to a higher layer in a Fog hierarchy, or to the Cloud itself for further processing [7, 12].

Applications are usually composed of (dependent) tasks. The scheduling of tasks using C2T resources is much more challenging than that of tasks on grids (confined systems) [13, 14, 15, 16] and on Clouds (more homogeneous systems) [17, 18, 19, 20, 21, 22] due to the considerable heterogeneity of both demands of applications and the capacity of the devices. Consequently, there is a need for schedulers to analyze various parameters before making decisions as to where tasks and virtualized resources should be run, including consideration of the availability of resources and their cost [7].

It is thus crucial for the efficient provisioning of services that the demands of applications arriving at the edge of the network be well understood and classified so that resources can be assigned for their processing. The mapping of applications onto Class of Services (CoS) should facilitate the matching between task requirements and resources, since labeling these tasks removes the burden of the analysis of the application requirements by the scheduler. Without a precise classification, the scheduling of application tasks and the allocation of resources can be less than optimal due to the complexity in dealing with the diversity of QoS and resource requirements. Mapping applications onto Class of Service is typical in communications network technologies that support QoS, such as LTE, 5G [23], and ATM [24] networks, and it is a key element for network providers to be able to offer different grades of service. Moreover, it is essential for the use of efficient classification algorithms to deal with the specific characteristics of C2T.

Techniques for the classification of network traffic have been extensively studied for the past few decades, especially for the provisioning of secure network services [25, 26]. However, very little attention has been paid to the classification of the demands and requirements of applications and services over the Internet [27]. Moreover, device mobility and the IoT have introduced new applications to the Internet, and these applications have not been considered in any of the classification schemes. This paper contributes with the definition of a set of Classes of Service for Fog computing, that takes into consideration the QoS requirements of the most relevant Fog applications, thus allowing the differentiation of the demands of a broad spectrum of applications.

The original contribution of this paper is a methodology for the classification of applications as to Class of Service, which considers their QoS requirements. This methodology can be used to design effective classifiers with an output that can facilitate the job of schedulers of applications tasks. In the scenario assumed in this paper, users subscribe directly or indirectly to fog infrastructure services. The first packet of a flow contains the QoS requirements of the application generating the packet flow. The proposed classifier will then map this application into a CoS using the information provided in the first packet. The CoS can then be used by a fog task scheduler to schedule application tasks and allocate resource to these tasks. It is our best knowledge that no previous paper has addressed the classification of fog applications. A case study dealing with the classification of a dataset containing Fog application features illustrates the use of this methodology. This methodology constitutes a first step towards the definition of a QoS provisioning framework to facilitate the definition of new business models in Fog Computing.

The rest of this paper is structured as follows. Section 2 overviews related work. Section 3 proposes a set of classes of service for Fog Computing, and provides a mapping between the recommended classes of service and the layers of the reference architecture presented by the OpenFog Consortium. Section 4 introduces the use of a typical machine learning-based methodology for Fog computing. Section 5 illustrates the implementation of the methodology described in Section 4 with a case study, which includes two scenarios, differing by the place where noise is introduced (either in the training set or in the testing set). Finally, Section 6 concludes the paper and points out directions for future research.

2. Related work

Different studies have analyzed application requirements to develop service models for both Cloud and Fog Computing. In Cloud computing, these studies have focused on Service Level Agreements (SLA) [28, 29, 30, 31] and Quality of Experience (QoE) management [32], while in Fog Computing, investigations have emphasized processing and analytics for specific applications [33], scheduling of applications to resources [34], resource estimation [35] and allocation [36, 37] for the processing of applications, and service placement [38, 39]. Next, these studies are briefly described.

Alhamad *et al.* [28] presented nonfunctional requirements of Cloud consumers, and defined the most important criteria for the definition and negotiation of SLAs between consumers and Cloud service providers.

Wu *et al.* [29] developed a Software as a Service (SaaS) broker for SLA negotiation. The aim was to achieve the required service efficiently when negotiating with multiple providers. The proposal involved the design of counter offer strategies and decision-making heuristics which considered time, market constraints and trade-off between QoS parameters. Results demonstrated that the proposed approach increases by 50% the profit and by 60% the customer satisfaction level.

Emeakaroha *et al.* [30] presented an approach for mapping SLA requirements to resource availability, called LoM2Hi, which is capable of detecting future SLA violations based on predefined thresholds to avert these violattions.

Emeakaroha *et al.* [31] proposed an architecture for application monitoring architecture, named Cloud Application SLA Violation Detection architecture (CASVid). CASViD monitors and detects SLA violations at the application layer, and includes tools for resource allocation, scheduling, and deployment. Results showed that the proposed architecture is efficient in monitoring and detecting situations of a single SLA violation.

Hobfeld *et al.* [32] discussed the challenges for QoE provisioning for Cloud applications with emphasis on multimedia applications. The authors also presented a QoE-based classification scheme of Cloud applications aligned to the end-user experience and usage domain.

Yang [33] investigated common components of IoT systems such as stream analytics, event monitoring, networked control, and real-time mobile crowdsourcing, for defining an architecture for Fog data streaming.

Cardellini *et al.* [34] modified the Storm data stream processing system (DSP) to operate in a geographically distributed and highly variable environment. To demonstrate the effectiveness of the extended Storm system, the authors implemented a distributed QoS aware scheduling algorithm for placing DSP applications near to the data sources and the final consumers. The main limitation of this study is the instability of the scheduling algorithm that affects negatively the application availability. Results showed that the distributed QoS-aware scheduler outperforms the default centralized one, improving the application performance.

Aazam *et al.* [35] developed a method, called MEdia FOg Resource Estimation (MeFoRE), to provide resource estimation on the basis of service give-up ratio, the record of resource usage and the required quality of service. The aim was to avoid resource underutilization and enhance QoS provisioning. MeFoRE methodology uses real IoT traces and traces of Amazon EC2 service.

Wang *et al.* [36] presented an edge architecture, called mobile micro-Cloud, to provide situational awareness to processing elements. The authors introduced an approach for consistent representation of application requirements for deployment in the mobile micro-Cloud environment.

He *et al.* [37] introduced QoE model which included user oriented metrics such as the Mean Opinion Score (MOS) and content popularity as well as the cost of cache allocation and transmission rate. The computed QoE value was used in a resource allocation problem formulated as a maximization problem solved by a shortest path tree algorithm. Results showed the benefit of using dynamic allocation to achieve high QoE values.

Mahmud *et al.* [38] proposed a QoE-aware application placement policy that prioritizes placement requests according to user expectation and the Fog available capacity. Two fuzzy logic models were employed to map applications to resources. Requests for application placement consider metrics such as service access rate, required resources and expected processing time. A linear optimization problem ensures that prioritized requests for application placement are mapped to Fog resources so that user QoE is maximized. Results indicated that the policy significantly reduces the processing time, resource availability, and the quality of service.

Skarlat *et al.* [39] evaluated the placement of IoT services on Fog resources, taking into account QoS requirements. The authors proposed an approach for the optimal sharing of resources among IoT services by employing a formal model for Fog systems. The authors introduced the Fog Service Placement Problem (FSPP) for placing IoT services on virtualized Fog resources while taking into account constraints such as execution time deadlines. Results showed that the proposed optimization model prevents QoS violations and decreases the execution cost when compared to a purely Cloud-based solution.

The classification methodology introduced in this paper differs from the aforementioned proposals by the definition of a set of Class of Service for Fog Computing and the use of machine learning algorithms to map applications onto these classes. To our knowledge, this is the first study that introduces a machine learning classification methodology to discriminate Fog Computing applications on the basis of QoS requirements. It is crucial for the efficient provisioning of services that the demands of applications arriving at the edge of the network be classified so that resources can be assigned for their processing. The related work in Fog Computing reported above concentrates on resource allocation and scheduling of applications. Most of the decisions on resource allocation and scheduling in those papers is limited to information on resource consumption by the applications. They do not consider several QoS requirements as done in the present manuscript. Moreover, no previous paper has addressed the classification of applications on the Cloud to Things (C2T) continuum. Most of the work dealing with SLAs and QoS/QoE considers only the Cloud. The Fog layers in C2T will increase the capacity of the system to support new applications, especially those with real-time constraints, which are not possible to be handled by the Cloud.

3. Classes of Service for Fog Applications

Fog Computing enables new applications, especially those with strict latency constraints and those involving mobility. These new applications will have heterogeneous QoS requirements and will demand Fog management mechanisms to cope efficiently with that heterogeneity. Thus, resource management in Fog computing is quite challenging, calling for integrated mechanisms capable of dynamically adapting the allocation of resources. A very first step in resource management is to separate incoming flow of requests into Classes of Service (CoS) according to their QoS requirements.

Bandwidth. Some applications request a minimally guaranteed throughput, i.e., a Guaranteed Bit Rate (GBR). Multimedia applications are bandwidth sensitive, although some of them use adaptive coding techniques to encode digitized voice or video at a rate that matches the currently available bandwidth.

Delay sensitivity. Some applications involve a specific latency threshold, below which latency must be assured, especially for real-time applications.

Loss sensitivity indicates the proportion of packets which does not reach their destination.

Reliability is concerned with the ability of the Fog components to carry out the desired operation in the presence of many types of failure. Some applications need to have failed Fog components quickly reestablished so that tasks can be performed within some latency bounds.

Availability provides a measure of how often the resources of the Fog are accessible to end-users. High availability is needed by applications and services that must be running all the time, such as mission-critical applications.

Security refers to the design and implementation of authentication and authorization techniques to protect personal and critical information generated by end users.

Data location indicates where the application data should be stored. Data can be stored locally, at the end device itself; near, or at a Fog node, or in a remote repository, in the Cloud. Requirements of data location for an application depend on factors such as response time constraints, the computational capacity of each Fog layer, and available capacity on network links.

Mobility is an intrinsic characteristic of many edge devices. Continuity of the offered services should be ensured, even for highly mobile end-users. Continuous connectivity is essential for the processing needed.

Scalability is related to the capability of an application to operate efficiently, even in the presence of an increasing number of requests from end users. The number of users in a Fog can fluctuate due to the mobility of the users, as well as the activation of applications or sensors. Streams of data in big data processing may need to be processed within a specific time frame. The demand on Fog nodes can fluctuate and resource elasticity needs to be provided to cope with these demands.

The mapping of applications into a set of classes of service is the first step in the creation of a resource management system capable of coping with the heterogeneity of Fog applications. This paper proposes various classes of service for Fog computing: Mission-critical, Real-time, Interactive, Conversational, Streaming, CPU-bound, and Best-effort. These classes will be defined and the typical applications using these classes identified.

The first CoS to be discussed is the *Mission-critical (MC)* class. It comprises applications with a low event to action time-bound, regulatory compliance, military-grade security, privacy, and applications in which a component failure would cause a significant increase in the safety risk for people and the environment. Applications include healthcare and hospital systems, medical localization, healthcare robotics, criminal justice, drone operations, industrial control, financial transactions, ATM banking systems, and military and emergency operations.

The *Real-time (RT)* class, on the other hand, groups applications requiring tight timing constraints in conjunction with effective data delivery. In this case, the speed of response in real-time applications is critical, since data are processed at the same time they are generated. In addition to being delay sensitive, real-time applications often require a minimum transmission rate and can tolerate a certain amount of data loss. This real-time class includes applications such as online gaming, virtual reality, and augmented reality.

The third class is denominated *Interactive (IN)*. In this case, responsiveness is critical, the time between when the user requests and actions manifested at the client being less than a few seconds. Moreover, users of interactive applications can be end devices or individuals. Examples of applications belonging to this class are interactive television, web browsing, database retrieval, server access, automatic database inquiries by tele-machines, pooling for measurement collection, and some IoT deployments.

The fourth class is the *Conversational (CO)* class. These applications include some of the video and Voice-over-IP (VoIP). They are characterized by being delay-sensitive but loss-tolerant with delays less than 150 milliseconds being perceived by humans, delays between 150 and 400 milliseconds can be acceptable, and those exceeding 400 milliseconds resulting in completely unintelligible voice conversations. On the other hand, conversational multimedia applications are loss-tolerant with occasional losses causing only occasional glitches in audio or video playback, and these losses can often be partially or fully concealed [40].

The fifth class of service is *Streaming (ST)*, which releases the user to download entire files, although in potentially long delays are incurred, before playout begins. Streaming applications are accessed by users on demand and must guarantee interactivity and continuous playout to the user. For this reason, the most critical performance measure for streaming video is average throughput [40]. Additionally, streaming can refer to stored or live content. In both cases, the network must provide each flow with an average throughput that is larger than the content consumption rate. In live transmissions, the delay can also be an issue, although the timing constraints are much less stringent than those of conversational voice. Thus, delays of up to ten seconds or so from when the user chooses to view a live transmission to when playout begins can be tolerated. Examples of streaming applications are high-definition movies, video (one-way), streaming music, and live radio and television transmissions.

The sixth class is *CPU-Bound (CB)* class which is used by applications involving complex processing models, such as those in decision making, which may demand hours, days, or even months of processing. Face recognition, animation rendering, speech processing, and distributed camera networks, are examples of CPU-Bound applications.

The final class is that of *Best-Effort (BE)*. It is dedicated to traditional best-effort applications over the Internet. For Best-effort applications, long delays are annoying but not particularly harmful; the completeness and integrity of the transferred data, however, are of paramount importance. Some examples of the Best-Effort class are e-mail downloads, chats, SMS delivery, FTP, P2P file sharing, and M2M communication.

Table 1 presents the relationship between the applications supported by Fog computing and the requirements of the classes of service explained above. The first column shows the recommended priority level of each class for potential adoption in scheduling systems.

Table 1: Class of Service and their requirements

Allocation Priority	**Class of Service**	**Service Quality Requirements**																					**Applications**
		Bandwidth	**Reliability**			**Security**			**Data storage**			**Data location**			**Mobility**			**Scalability**			**Delay sensitivity**	**Loss sensitivity**	
			Low	**Important**	**Critical**	**Low**	**Medium**	**High**	**Transient**	**Short duration**	**Long duration**	**Local**	**Vicinity**	**Remote**	**Low**	**Medium**	**High**	**Low**	**Medium**	**High**			
1	MC	GBR			✓			✓	✓			✓			✓	✓	✓			✓	Yes	Yes	Healthcare, criminal justice, financial, biological traits, residence and geographic, military, emergency.
2	RT	GBR		✓			✓		✓			✓			✓			✓			Yes	No	Online gaming, IoT deployments, industrial control, virtual and augmented reality, interactive television, telemetry.
3	IN	GBR	✓					✓	✓				✓		✓					✓	Yes	Yes	Interactive television, object hyperlinking, web browsing, database retrieval, server access, automatic database enquiries by tele-machines, pooling for measurements collection, and some IoT deployments.
4	CO	GBR		✓			✓		✓				✓				✓			✓	Yes	No	Voice messaging, VoIP, videoconference.
5	ST	GBR		✓			✓				✓			✓		✓	✓		✓		Yes	No	Internet radio, video (one-way), high quality streaming audio.
6	CB	GBR		✓				✓		✓	✓			✓	✓				✓		Yes	Yes	Face recognition, animation rendering, speech processing, distributed camera networks.
7	BE	NGBR	✓			✓					✓			✓	✓					✓	No	Yes	Network signaling, all non-critical traffic such as TCP-based data: www, e-mail, chat, FTP, P2P file sharing, progressive video and other miscellaneous traffic.

Table 2 shows the range of QoS requirement values for each class of service: Bandwidth [40, 32], Reliability [41], Security [42], Data storage [28, 32], Data location [28, 32, 41], Mobility [41], Scalability [28, 32, 41], Delay sensitivity [7, 32], Loss sensitivity [43]. These ranges are used to generate the synthetic dataset of Fog applications and are employed for training and testing samples to evaluate the classifiers in this paper (Section 5).

Table 2: Intervals of the QoS requirements for Fog computing.

QoS Requirements	**Nominal Categories**	**Intervals**	**Class of Service**						
			MC	**RT**	**IN**	**CO**	**ST**	**CB**	**BE**
Bandwidth (Mbps)	Low	$0 < x \leqslant 1$	✓		✓	✓			✓
	Medium	$1 < x \leqslant 5$					✓	✓	
	High	$5 < x \leqslant 1000$		✓					
Reliability	Low	$x = 1$			✓				✓
	Important	$x = 2$		✓		✓	✓	✓	
	Critical	$x = 3$	✓						
Security	Low	$x = 1$							✓
	Medium	$x = 2$		✓		✓	✓		
	High	$x = 3$	✓		✓			✓	
Data storage (h)	Transient	$0 < x \leqslant 1$	✓	✓	✓	✓			
	Short duration	$1 < x \leqslant 730$						✓	
	Long duration	$730 < x \leqslant 8760$					✓	✓	✓
Data location (ms)	Local	$0 < x \leqslant 10$	✓	✓					
	Vicinity	$10 < x \leqslant 20$			✓	✓			
	Remote	$20 < x \leqslant 100$					✓	✓	✓
Mobility (Km/h)	Low	$0 < x \leqslant 5$	✓	✓	✓			✓	✓
	Medium	$5 < x \leqslant 25$	✓				✓		
	High	$25 < x \leqslant 100$	✓			✓	✓		
Scalability (No. of IoT users/ end users)	Low	$0 < x \leqslant 60$		✓					
	Medium	$60 < x \leqslant 120$					✓	✓	
	High	$120 < x \leqslant 200$	✓		✓	✓			✓
Delay sensitivity (Interaction latency in ms)	Low	$1000 < x \leqslant 100000$					✓	✓	✓
	Moderate	$10 < x \leqslant 1000$			✓	✓			
	High	$0 < x \leqslant 10$	✓	✓					
Loss sensitivity (PELR)	Low	$10^{-3} < x \leqslant 10^{-2}$				✓			
	Moderate	$10^{-6} < x \leqslant 10^{-3}$		✓			✓		
	High	$0 < x \leqslant 10^{-6}$	✓		✓			✓	✓

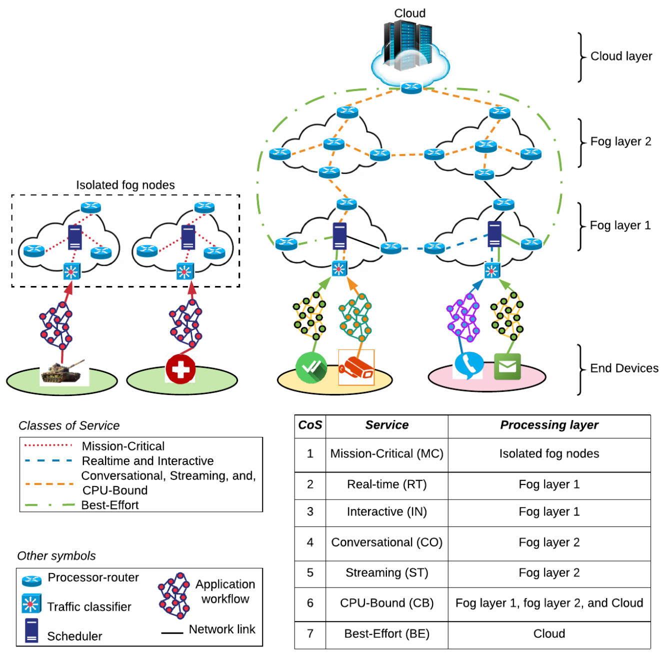
The reference architecture proposed by the OpenFog consortium in [2] provides a structural model for Fog-based computation on several tiers of nodes. The tiers differ in relation to the amount and type of work which can be processed on them, the number of sensors, the capacity of the nodes, the latency between nodes, reliability, and availability of nodes. Nodes at the edge are involved in sensor data acquisition/collection, data normalization, and command/control of sensors and actuators, while nodes that are closer to the Cloud aggregate and transform data into knowledge. As one moves further away from the edge, the overall intelligence and capacity of the system increase.

Fig. 1 presents a distributed multi-layer architecture, based on the OpenFog reference architecture, which is composed of four layers: the Cloud, at the top, a layer of end devices at the bottom and two intermediate Fog layers. Fig. 1 also provides a mapping between the proposed classes of service and a multi-layer Fog-Cloud architecture. The bottom layer, composed of IoT and end-devices, sends application requests to the classifier, located on the first Fog layer. An application request is composed of the workflow of tasks and their demands, as well as the QoS requirements for the application. The classifier identifies the CoS of the application and forwards it to the scheduler, which decides where the application should be processed, whether on the first Fog layer, on the second Fog layer, or in the Cloud.

Not all layers are involved in the processing of all tasks. Since Real-time, Interactive, Conversational, and Streaming applications, such as online sensing, object hyperlinking, video conferencing, and stored streaming are delay-sensitive, these applications must be processed as close as possible to the end user, preferably at nodes located on the first and second Fog layer. CPU-bound applications require many processing resources, and for this reason, can involve all the layers of the reference architecture for the processing of tasks. Best-effort applications, such as e-mails, can be processed in the Cloud since there are no delay constraints for this class.

The possibility of having a hierarchical layered system is one of the significant differences between Fog computing and edge computing. Edge computing is mainly concerned in bringing the computation facilities closer to the user; however, in a flat non-hierarchical architecture [44]. A layered architecture can introduce additional communication overhead for processing tasks at different layers. However, it has been shown that if the scheduling of tasks and resource reservations are properly carried out, processing in a hierarchical architecture can reduce communication latency and task waiting time for processing when compared to a flat architecture [45].

In the scenario assumed in this paper, users subscribe directly or indirectly to Fog infrastructure services. The first packet of a flow contains the QoS requirements of the application generating the packet flow. The proposed classifier will then map this application into a CoS using the information provided in the first packet. Alternatively, the first packet could already carry the CoS of the application. However, such an option would make rigid the CoS adopted by the Fog provider, preventing the redefinition of this CoS for the handling of new applications with unique QoS requirements.



CoS	Service	Processing layer
1	Mission-Critical (MC)	Isolated fog nodes
2	Real-time (RT)	Fog layer 1
3	Interactive (IN)	Fog layer 1
4	Conversational (CO)	Fog layer 2
5	Streaming (ST)	Fog layer 2
6	CPU-Bound (CB)	Fog layer 1, fog layer 2, and Cloud
7	Best-Effort (BE)	Cloud

Figure 1: Fog computing architecture and Class of Service.

4. Classification methodology based on machine learning

This section introduces a methodology for choosing and evaluating classifiers for Fog computing applications. It provides a step-by-step procedure for grounded choices of classifiers. Indeed, this methodology can be easily modified for the classification of applications in networked systems.

Classification techniques based on ML aim at mapping a set of new input data to a set of discrete or continuous valued output. Fig. 2 summarizes the key steps in the building of a classifier of Fog applications based on ML algorithms. In this paper, the classification steps were executed offline. Indeed, the best performing classifier evaluated in these steps can be executed on-line in an operational Fog.

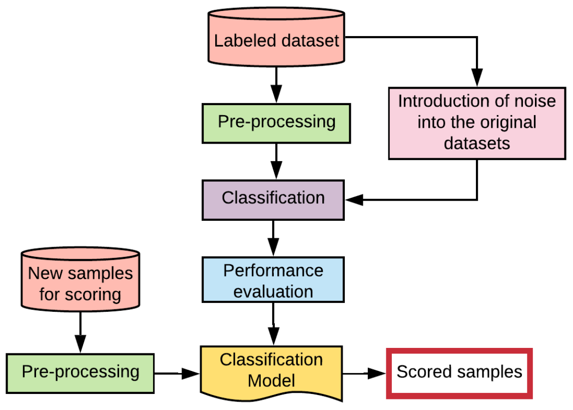


Figure 2: Typical ML-based classification methodology adopted in the context of Fog computing networks.

The first step is the creation of a *labeled dataset* containing QoS attributes of Fog applications, which can be either real or synthetic. A real dataset is one collected from a system in operation while a synthetic one involves data collection generated by models. Real-world datasets usually contain sensitive data [46] and are often unavailable for the maintenance of the user information. Thus, the use of synthetic data sets is quite common, especially in the studies of systems yet to be built.

Since the value of QoS attributes differs widely, these values should be *pre-processed* to produce compatible ranges of values for classification. Pre-processing includes operations for data transformation, which standardize and consolidate data into more appropriate forms for classification, while data reduction includes the selection and extraction of both features and examples in a database [47, 48]. Data normalization avoids the handling of an attribute which has large values that dominate the results of the classification, thus improving the predictive power of the model. Feature selection, on the other hand, removes redundant and irrelevant data from the input of the classifier, without compromising critical attribute information [48].

Noise is an unavoidable problem in collecting data from real-world systems. It can change the knowledge extracted from the data set and affects the accuracy of the classification, building time, size, and interpretability of the classifier [27, 49]. Common sources of noise are channel capacity fluctuation, fluctuation in the availability of computational resources, imprecision inherited from measurement tools, and the inability to accurately estimate the true demands of applications.

In such noisy scenarios, robustness is considered more important than performance because robustness allows a priori knowledge of the behavior expected from a learning method despite noise when it is unknown [47]. Robustness [50] is defined as the capability of an algorithm to be insensitive to data corruption and, consequently, more resilient against the impact of noise. A copy of the original dataset should be contaminated by the *introduction of noise* at different levels to check the robustness of a classifier. In this paper, uniform attribute noise levels of 10%, 30%, and 50% are employed. The performance of the classifiers which learned from the original data set is compared to that of those which learned from a noisy data set. The most robust classifiers are those which learned from noisy data sets yet produced results similar to those learned from a noise-free data set [47].

Classification techniques based on ML can then be applied. The performance of the classifier should be assessed by a *performance evaluation process*, which encompasses both the measurement of performance and the result of statistical tests. The adequacy of performance is usually assessed by metrics such as accuracy, efficiency, and robustness. Statistical testing gathers evidence of the extent to which an evaluation metric on the resampled data sets is representative of the general behavior of the classifier [51].

At this point, the *classification model* is ready to receive new input for scoring. The new data, however, must also be subjected to a *pre-processing* process.

5. Classification of Fog applications

This section illustrates step by step the methodology presented in the previous section for the classification of Fog applications using the CoS presented previously. Moreover, an example of a Decision Tree that classifies Fog computing applications from the values of their QoS requirements is provided at the end of this section.

5.1. Labeled dataset

To train and test the classifiers employed in this paper, we built a dataset [1] composed of 14,000 mutually exclusive applications generated from data in

[1]publicly available at http://bit.ly/34x6X1O

the intervals of values acceptable for each QoS requirement of the application. 90% of the data were reserved for training, while the remaining 10% were used for testing. It was assumed that each incoming application had additional fields containing nine QoS requirements, from now on referred to as "attributes": Bandwidth, Reliability, Security, Data Storage, Data location, Mobility, Scalability, Delay sensitivity, and Loss sensitivity.

Attribute values were assigned by employing a uniform probability distribution, within the intervals specified for each CoS in Table 2.An independent random number generator randomly created the values of each attribute. Transient data were removed according to the Moving Average of Independent Replications procedure [52]. Attribute values were made up of safe and borderline examples. Safe examples were placed in relatively homogeneous areas concerning the class label. Borderline examples, on the other hand, are located in the area surrounding class boundaries, where different classes overlap. Also, to estimate the robustness of the classifiers, the third group of attribute values, called noisy examples, was generated. The term noisy sample will be used in this paper to refer to the samples generated to represent the corruption of their attribute values.

Fig. 3 illustrates the safe samples, labeled as S, the borderline examples, labeled as B, and thenoisy samples, labeled as N. The continuous line shows the decision boundary between the two classes.

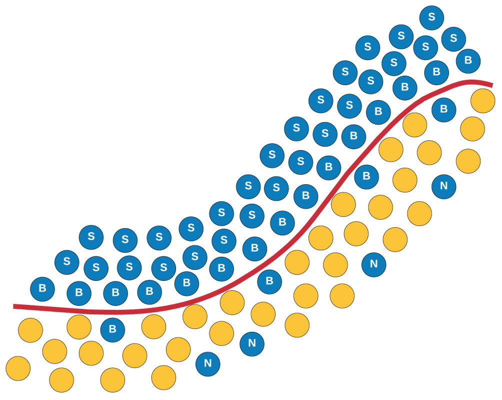


Figure 3: The three types of samples considered in this paper: safe (S), borderline (B), and noisy (N).

5.2. Pre-processing

Z-score normalization was used to adjust attribute values defined on a different scale. Mean and standard deviation were computed on the training set, and then, the same mean and standard deviation were then used to normalize the testing set.

We reduced the number of input attributes to be used by classification algorithms. This process, known as dimensionality reduction, removes irrelevant, redundant, and noisy information from the data, often leading to enhanced performance in learning and classification tasks [53]. Two techniques can be used for dimensionality reduction: one, by using future selection techniques such as Relief-F [54], CFS [55], MCFS [56], and the Student's t-test, which rank the given feature set so that the least significant features can be removed from the problem. The second way involves feature extraction techniques, such as Principal Components Analysis (PCA), which creates new features from the given feature set. The resulting number of the features is less than that the initial set of features.

In this paper, the significance level of the impact of each input attribute on the system is determined utilizing a PCA, guided further by a correlation analysis and the semantics of the Fog computing environment. The correlation analysis and complementarity with the PCA are explained below.

The correlation analysis is a statistical evaluation technique used to study the strength of a dependence between pairs of attributes. Fig. 4 provides a graphic representation of the correlation matrix among dataset attributes for Fog applications.

The correlation matrix shows that there is a statistical association of more than 50% between the following variables: "Data storage" and "Data location" (0.623), "Data storage" and "Delay sensitivity" (0.596), "Loss sensitivity" and "Mobility" (0.563), "Bandwidth" and "Scalability" (-0,605) and, "Data location" and "Delay sensitivity" (0.674). The symbol "-"in the correlation value between the attributes of "Bandwidth"and "Scalability"indicates an inverse relationship between the two.

Principal components analysis (PCA) [57] is a common approach for dimensionality reduction that uses techniques from linear algebra to find new attributes, denominated principal components, which are linear combinations of the original attributes. They are orthogonal to each other, and capture the maximum amount of variation in the data. Fig. 5 shows the scree plot of the percent variability explained by each principal component.

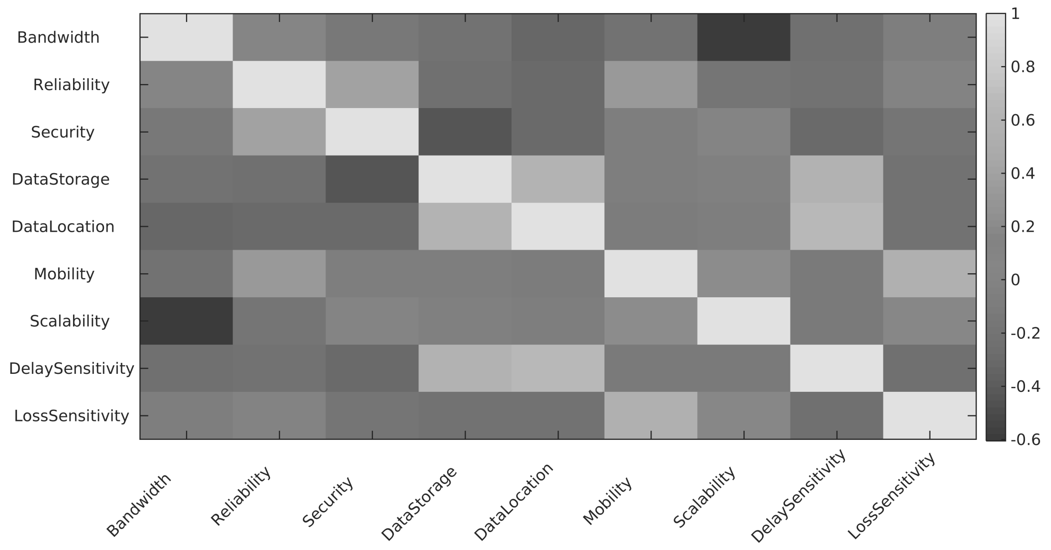


Figure 4: Attribute association estimates.

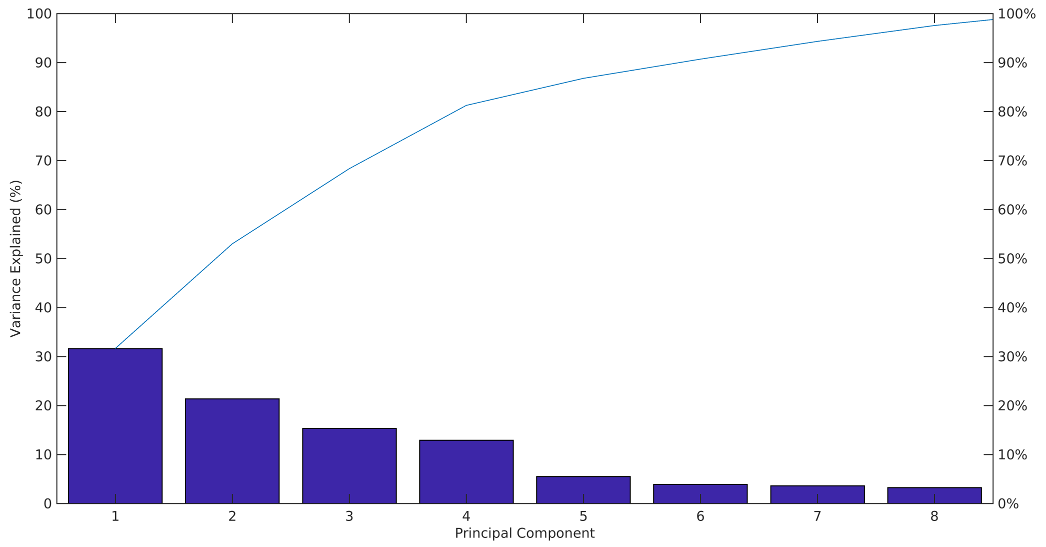


Figure 5: Scree plot of the percent variability explained by each principal component.

As illustrated by Fig. 5, the first seven principal components explain 94.314% of the total variance. The first component by itself explained less than 35% of the variance, so more components might be needed. Also, Fig. 5 reveals that the first three principal components explain roughly two-thirds of the total variability in the standardized ratings.

In addition to the percent variability explained by each principal component, all nine attributes were represented in a bi-plot by a vector. The direction and length of the vector indicate the contribution of each attribute to the two principal components in the plot. For instance, Fig. 6 shows the coefficients of each attribute concerning the first two principal components.

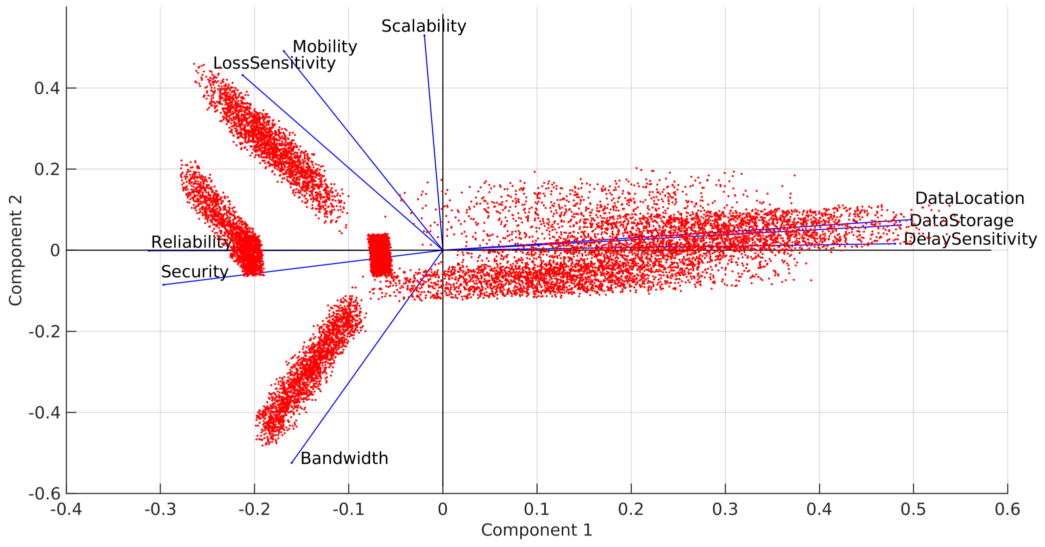


Figure 6: Orthonormal principal component coefficients for each variable and principal component scored for each observation (principal components 1 and 2).

The other five principal components were also plotted in bi-plots. Table 3 shows the contribution of the attributes to each principal component.

Interpretation of the principal components is based on finding which variables are most strongly correlated with each component, that is, which of these numbers is large, the farthest from zero in either direction. The decision as to what values should be considered large is a subjective one and reflects knowledge of the system under evaluation. It was determined that a correlation value was relevant to our study when it was above 0.48, since this value is the largest within each principal component, and most of the variables having this high value are highly correlated, as shown by the components of the correlation matrix. These large correlation values are in boldface in Table 3.

The principal component results can be interpreted with respect to the

Table 3: Attribute coefficients for each principal component.

Attribute	**Principal Component**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Bandwidth	-0.161	**-0.524**	-0.449	-0.065	0.079	0.118	0.114
Reliability	-0.312	-0.001	0.040	**0.702**	0.313	0.295	-0.318
Security	-0.297	-0.085	**0.595**	0.283	-0.342	-0.286	-0.021
Data Storage	**0.486**	0.062	-0.099	0.181	**0.490**	-0.375	-0.389
Data location	**0.497**	0.075	0.022	0.213	-0.351	-0.283	0.062
Mobility	-0.169	**0.491**	-0.327	0.350	0.117	-0.207	**0.645**
Scalability	-0.020	**0.530**	0.337	-0.366	0.309	0.318	-0.014
Delay sensitivity	**0.480**	0.016	0.016	0.295	-0.273	**0.673**	0.134
Loss sensitivity	-0.213	0.432	-0.460	-0.044	**-0.481**	0.011	**-0.544**

value deemed to be significant. The first principal component correlated strongly with three of the original attributes. Thus, the first principal component increases with increasing Data storage, Data location, and Delay sensitivity, suggesting that these three attributes vary together.

On the other hand, the coefficients belonging to the second principal component show that the behavior of the feature bandwidth opposes that of the behavior of the features Mobility and Scalability. This reflects the fact that greater mobility is associated with changes in the network topology, which, in turn, increases the fluctuations in communication links and reduces the bandwidth availability. Moreover, since bandwidth is a finite resource, if the number of users connected to the Fog increases the rate at which each user transmits and receives data decreases.

Other features that reveal an inverse relationship are Data Storage and Loss sensitivity, in the fifth principal component, and mobility and loss sensitivity in the seventh principal component.

Finally, the third, fourth, and sixth principal components increase with only one of the values, that is, there is only one variable with a value 0.48 or higher. These variables are Security, Reliability, and Delay sensitivity, respectively. Accordingly, the third, fourth, and sixth principal components can be interpreted as measures of how necessary the use of isolated nodes is to process the application, how quickly failed Fog components should be reestablished, and how sensitive the Fog application is to the delay.

Based on the analysis described above and considering the semantics of the case study about which variables deserve more attention into the Fog computing environment, redundant attributes such as Data location, Data storage, and Mobility have been removed. Thus, six of the original nine attributes were selected and maintained for the stage of classification. These were Delay sensitivity, Scalability, Loss sensitivity, Security, Reliability, and Bandwidth.

5.3. Classification

Seven classifiers are evaluated for potential adoption: Adaptive neuro-fuzzy inference system from data, using subtractive clustering (ANFIS), Decision Tree (DT) Artificial neural network with 2 hidden layers, trained with the Levenberg-Marquardt backpropagation algorithm, (ANN(1)); Artificial Neural Network with 1 hidden layer, trained with the algorithm Scaled conjugate gradient backpropagation (ANN(2)); Artificial Neural Network with 2 hidden layers, trained with the algorithm Scaled conjugate gradient backpropagation (ANN(3)); K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). These algorithms have different characteristics with respect to noise sensitivity, the speed of learning, and the speed of prediction. For instance, SVMs are known to be very accurate but also sensitive to noise [47]. ANNs predict rapidly, but the speed of learning is low, while KNN provides rapid learning, but are considered difficult to interpret [58]. These classifiers are typically employed in the context of complex recognition or prediction applications, especially when there is a lack of labeled samples, a lack of time for training and testing, or even a lack of appropriate hardware for timely assessment of the quality of classification models. These same conditions are present in typical Fog computing scenarios, as shown in Fig. 1, in which heterogeneous devices with limited computing and storage capabilities must interoperate in real time to ensure compliance with the QoS requirements of time-sensitive applications. The investigation of data-driven approaches (e.g., based on deep learning models) for constrained processing scenarios [59, 60, 61] as in our target application is left for future work.

5.4. Performance evaluation

This section focuses on two main subtasks of the evaluation process: measurements of the performance and statistical significance of the performance metrics. The hardware configuration used for both the dataset generation and the experiments carried out was: Intel Core i7 processor, 12GB of RAM, 1 TB of storage and Windows 10 operating system. First, the performance of each classifier is evaluated under ideal conditions, that is, without noise. Each classifier is assessed by measuring its accuracy and efficiency. Then, the performance of each classifier is assessed in the presence of noise. In this case, the robustness of each classifier in two different scenarios is assessed: when noise is introduced into the training set, and when noise is introduced in the testing set. In both cases, with and without noise, Wilcoxon's signed-rank test is employed to determine whether or not the difference in accuracy between two classifiers is statistically significant. Inserting noise in the training data set is designed to evaluate the robustness of the classifier trained by the specific data set. Inserting noise in the test data set aims at assessing the robustness of a classifier in the face of data sets with no evident data relationship, for which data relationships are not so evident.

A stratified *10*-Fold Cross-validation (*10*-FCV) [62] protocol was adopted. In this protocol, there are 10 partitions, each with the same proportion of samples belonging to each class. Nine of them are used for training, and the remaining fold is used for testing. This process is repeated 10 times, with each of the 10 folds used exactly once as the testing data. Finally, the 10 results obtained from each one of the test partitions are combined to produce a single average value representing accuracy. Table 4 summarizes the results obtained from the accuracy estimation for each classifier assessed without noise.

Table 4: Accuracy estimation results.

Classification Technique	**Testing**	
	Average Accuracy (%)	**Average Time (s)**
ANFIS	99.271	0.048
DT	99.986	0.029
ANN(1)	100	0.032
ANN(2)	100	0.035
ANN(3)	100	0.031
KNN	100	0.073
SVM	100	0.044

Table 4 shows that the average accuracy results for the testing process were close to 100%, except for the ANFIS was only 99.2% accurate. The shortest prediction time was obtained with the DT algorithm, while the KNN took the longest time. One possible explanation is that the DT algorithm divides the input space, matching the way the attributes were defined and assigned to each CoS by using intervals. The way the synthetic dataset was created may have led to application instances with properties which were easy to model, thus boosting the performance of the evaluated classifiers.

A two-tailed Wilcoxon signed-rank test with a significance level of 0.05 was also applied to the observations obtained from the different classifiers with no attribution of noise. The results revealed that the accuracy level of the classification obtained from the DT, ANN(1), ANN(2), ANN(3), KNN, and SVM were approximately the same. In contrast, observations obtained from the ANFIS reveled statistically significant differences in relation to the observations obtained from the other classifiers.

An attribute noise was introduced into the original dataset to check the effect of noise on the classifiers. Corrupting the data impacts directly on the significance of the attributes for the purpose of classification [47]. Moreover, attribute noise includes erroneous attribute values, which is one of the most common types of noise in real-world data [49].

Noise is introduced into each partition in a controlled manner, i.e., a certain percentage of the values of each attribute in the dataset were altered. To this end, the steps employed by the authors in [47] were followed. To corrupt each attribute A_i, a certain percentage of the examples in the dataset were chosen, and the A_i value of each was assigned a random uniform value from the domain D_i of the attribute A_i.

Noise was introduced into the training partitions to create a noisy data set from the original, as follows:

(i) Noise as a percentage of the original value was introduced into a copy of the full original dataset.

(ii) The two datasets, the original and the noisy copy, were partitioned into 10 equal folds, i.e., each with the same number of examples of each class.

(iii) The training partitions are built from the noisy copy, whereas the test partitions were formed from examples from the noise-free dataset.

Noise was introduced into the testing partitions following the same steps described above, except that in Step *iii* the testing partitions are built from the noisy copy, while the training partitions were formed from examples from the noise-free data set.

The methodology followed in this subsection presents two differences concerning the methodology described in [47]. First, the one proposed here includes the same number of examples of each class in the percentage of values of each attribute in the data set to be corrupted. Second, the introduction of attribute noise was extended to a second simulation scenario in which accuracy and robustness were estimated for each one of the individual classifiers using a clean training set and a testing set with attribute noise. Introducing attribute noise into the training set while maintaining the testing set clean enabled the assessment of the capacity of the classifier to deal with problems in the training stage, such as overfitting. Overfitting occurs when the model is too tightly adjusted to data offering high precision in known cases, but behaving poorly with unseen data. Conversely, introducing attribute noise into the testing set while maintaining the training set clean, enables us to assess the robustness of the trained model.

After introducing noise, the accuracy of classifiers is determined by means of 5 runs of a stratified 10-Fold Cross-Validation (FCV). Hence, a total of 50 runs per dataset, noise type, and level are averaged. Ten partitions make the noise effects more notable, since each partition has a large number of examples (1,400). The robustness of each algorithm is then estimated by using the Relative Loss of Accuracy (RLA) given by Equation 1 is:

$$RLA_{x\%} = \frac{Acc_{0\%} - Acc_{x\%}}{Acc_{0\%}}, \tag{1}$$

where $RLA_{x\%}$ is the relative loss of accuracy at a noise level of $x\%$. $Acc_{0\%}$ is the test accuracy in the original case, that is, with 0% of induced noise, and $Acc_{x\%}$ is the test accuracy with a noise level $x\%$.

Next, the robustness of the classifiers when the noise has been introduced for the two mentioned scenarios will be evaluated.

5.4.1. Classification using a training set with attribute noise and a clean testing set

Table 5 shows the average performance and robustness results for each classification algorithm at each noise level, from 0% to 50%, on training datasets with uniform attribute noise.

As can be observed in Table 5, the DT is the most robust classifier for all noise levels. On the other hand, the ANN(1), ANN(2), and ANN(3) present high robustness for noise levels (10-30%). Conversely, the RLA of classifiers based on neural networks rises linearly to 9% when the noise level is 50%. The least robust classifier is the ANFIS, for which the loss of accuracy increases exponentially as the proportion of noise level rises, to the point that when the noise level is 50%, its RLA is above 21% of that a clean dataset.

Table 5: Test accuracy and RLA results of classifiers trained with noisy datasets.

	Noise Level (%)	ANFIS	DT	ANN(1)	ANN(2)	ANN(3)	KNN	SVM
Test accuracy results	0	99.271	99.986	100.000	100.000	100.000	100.000	100.000
	10	94.429	99.986	99.911	99.950	99.921	99.800	99.997
	30	84.900	99.950	99.676	99.800	99.691	99.029	99.836
	50	78.564	99.979	90.807	91.370	91.419	99.193	99.600
RLA values results	0	-	-	-	-	-	-	-
	10	0.04878	0.00000	0.00089	0.00050	0.00079	0.00200	0.00003
	30	0.14477	0.00036	0.00324	0.00200	0.00309	0.00971	0.00164
	50	0.20859	0.00007	0.09193	0.08630	0.08581	0.00807	0.00400

Fig. 7 shows the accuracy ratio and testing time when training takes place with both clean datasets, and those disrupted by uniform attribute noise levels of 10%, 30%, and 50%. A marker identifies each classification algorithm, and a different color identifies each noise level. The light- bands indicate the areas of the greatest accuracy or the slowest testing times, and the light-purple intersection of these bands indicates the area where the best results for both accuracy and testing time are found.

The DT algorithm takes only 25 milliseconds for classification with the greatest accuracy for up to 1,400 applications simultaneously arriving at the edge, when training has taken place using datasets with a uniform attribute noise level of 50%.

Table 6 presents the results of a two-tailed Wilcoxon signed-rank test (considering a significance level of 0.05) to verify statistical differences between the accuracy results of classifiers trained with noisy datasets. Each cell shows the results of the statistical tests between a single classifier with the others for the four levels of noise *(nl)*. Left '$\leftarrow$'and up '$\uparrow$'arrows indicate the most accurate, while an empty cell refers to "no statistical difference between the pairs of classifiers"in that row and column.

Table 6 shows the effect of noise data sets in training. The performance of ANFIS is the least accurate. ANN(1), ANN(2), and SVM produce a similar performance while the DT outperform most of the results of the rest of classifiers when it is trained using datasets with an attribute noise level equal to or greater than 30%.

Table 6: Statistical test for the accuracy of classifiers trained with noisy datasets. nl denotes the percentage of noise level present in the training dataset.

	nl	**ANFIS**	**DT**	**ANN(1)**	**ANN(2)**	**ANN(3)**	**KNN**	**SVM**
ANFIS	0	–	↑	↑	↑	↑	↑	↑
	10	–	↑	↑	↑	↑	↑	↑
	30	–	↑	↑	↑	↑	↑	↑
	50	–	↑	–	–	–	↑	↑
DT	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	←	–
	30	←	–	←	←	←	←	←
	50	←	–	←	←	←	←	←
ANN(1)	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	–	–
	30	←	↑	–	–	–	←	–
	50	–	↑	–	–	–	–	–
ANN(2)	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	←	–
	30	←	↑	–	–	–	←	–
	50	–	↑	–	–	–	–	–
ANN(3)	0	←	–	–	–	–	–	–
	10	←	–	–	–	–	–	↑
	30	←	↑	–	–	–	←	↑
	50	–	↑	–	–	–	–	–
KNN	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	↑
	30	←	↑	↑	↑	↑	–	↑
	50	←	↑	–	–	–	–	↑
SVM	0	←	–	–	–	–	–	–
	10	←	–	–	–	←	←	–
	30	←	↑	–	–	←	←	–
	50	←	↑	–	–	–	←	–

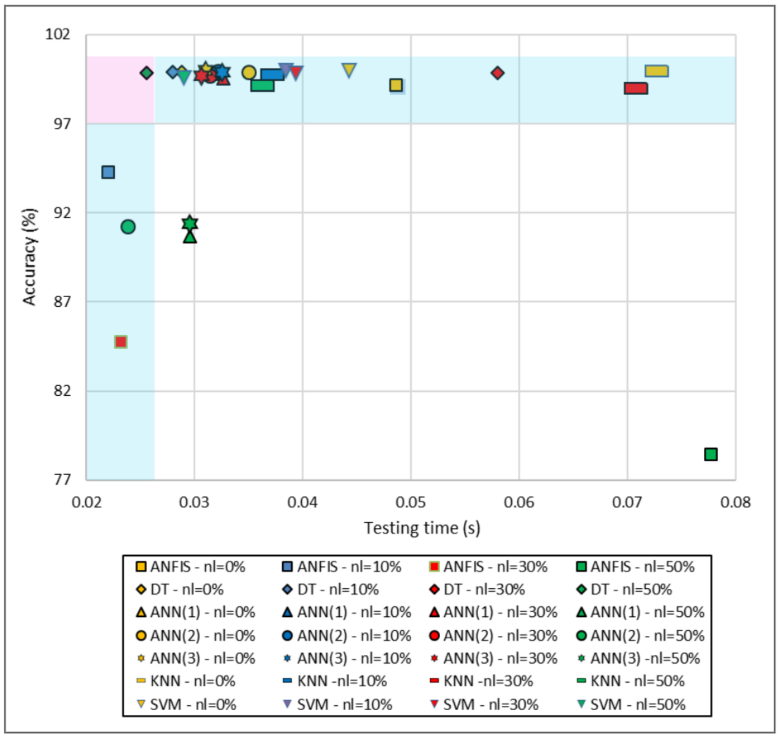


Figure 7: Accuracy rates concerning the testing time for classifiers trained with both clean and noisy datasets.

5.4.2. Classification using a clean training set and a testing set with attribute noise

Table 7 shows the results for average performance and robustness for each classification algorithm at each noise level, from 0% to 50%, from the testing of datasets with uniform attribute noise.

As evinced in Table 7, for all classifiers, accuracy decreases exponentially with an increase in the noise level of the testing dataset. In this situation, the most robust classifiers are ANN(2), DT, ANN(3), ANN(1), and KNN.

Fig. 8 illustrates the results of the classification algorithms when both accuracy and testing time are considered with both clean and noise datasets (levels of 10%, 30%, and 50%). A marker identifies each classification algorithm, and a different color identifies each noise level. The light-blue bands indicate the areas of the greatest accuracy rates or the slowest testing times, and the light-purple intersection of these bands indicates the area where the best results were obtained when considering both accuracy and testing time.

Table 7: Test accuracy and RLA results of classifiers tested with noisy datasets.

	Noise Level (%)	**ANFIS**	**DT**	**ANN(1)**	**ANN(2)**	**ANN(3)**	**KNN**	**SVM**
Test accuracy results	0	99.271	99.986	100.000	100.000	100.000	100.000	100.000
	10	72.214	85.693	83.570	86.131	84.031	84.093	81.79
	30	41.343	63.114	59.464	63.860	60.467	59.971	55.403
	50	27.414	47.264	45.381	48.564	46.176	44.764	40.799
RLA values results	0	-	-	-	-	-	-	-
	10	0.273	0.143	0.164	0.139	0.160	0.159	0.189
	30	0.584	0.369	0.405	0.361	0.395	0.400	0.446
	50	0.724	0.527	0.546	0.514	0.538	0.552	0.592

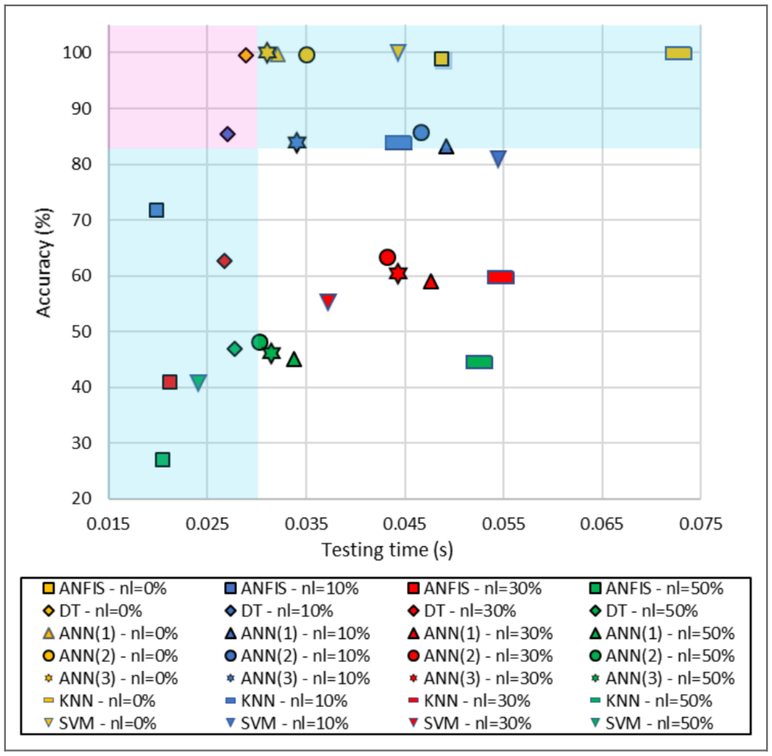


Figure 8: Accuracy rates concerning the testing time for classifiers tested with both clean and noisy datasets.

The DT algorithm takes less than 30 milliseconds for classification for with the greatest level of accuracy, up to 1,400 applications simultaneously arriving at the edge, when the input is a noise dataset as long as the noise level does not exceed 10%.

Table 8 presents the results of a two-tailed Wilcoxon signed-rank tests (considering a significance level of 0.05) to verify the statistical differences between the accuracy of the different classifiers when tested with noisy datasets. Each cell shows the result of the statistical test between the pairs of classifiers with different percentages of noise (nl). Left '←'and up '↑'arrows indicate the greatest accuracy while an empty cell refers to "no statistical difference between the pair of classifiers"in that row and column.

Table 8: Statistical test for the accuracy of classifiers tested with noisy datasets. nl denotes the percentage of noise level present in the testing dataset.

	nl	**ANFIS**	**DT**	**ANN(1)**	**ANN(2)**	**ANN(3)**	**KNN**	**SVM**
ANFIS	0	–	↑	↑	↑	↑	↑	↑
	10	–	↑	↑	↑	↑	↑	↑
	30	–	↑	↑	↑	↑	↑	↑
	50	–	↑	↑	↑	↑	↑	↑
DT	0	←	–	–	–	–	–	–
	10	←	–	←	–	←	←	←
	30	←	–	←	–	←	←	←
	50	←	–	–	–	–	←	←
ANN(1)	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	←
	30	←	↑	–	↑	–	–	←
	50	←	–	–	↑	–	–	←
ANN(2)	0	←	–	–	–	–	–	–
	10	←	–	←	–	←	←	←
	30	←	–	←	–	←	←	←
	50	←	–	←	–	–	←	←
ANN(3)	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	←
	30	←	↑	–	↑	–	–	←
	50	←	–	–	–	–	–	←
KNN	0	←	–	–	–	–	–	–
	10	←	↑	–	↑	–	–	←
	30	←	↑	–	↑	–	–	←
	50	←	↑	–	↑	–	–	←
SVM	0	←	–	–	–	–	–	–
	10	←	↑	↑	↑	↑	↑	–
	30	←	↑	↑	↑	↑	↑	–
	50	←	↑	↑	↑	↑	↑	–

The effect of noisy data set in testing is shown in Table 8. The performance of ANFIS is the least accurate. Moreover, the DT produces the most accurate classification results in classifications independent of the presence of noise.

5.5. Classification model

The final step of the proposed methodology is the selection of a classifier. The results indicate that the DT was the most accurate and robust classifier.

The decision tree algorithm does not need to assess all the attributes to classify an application since various services have exclusive features. For example, mission-critical applications are the only ones for which reliability takes on a “critical”value. Therefore, assessing certain features makes classification a more efficient process. This is an attractive characteristic which makes Decision Tree an ideal algorithm for the classification of applications in Fog computing. Moreover, the Decision Tree algorithm is easy to interpret, fast for fitting and prediction, and does not use much memory. Given these characteristics, the Decision Tree algorithm can be run by devices such as routers, switches, and servers, located on the first Fog layer of the reference architecture introduced by the OpenFog Consortium in [2]. After classification, the output of the classifier serves as input for the scheduler, also located at the first Fog layer, which decides where the application should be processed.

6. Conclusions

This paper has introduced the use of ML classification algorithms as a tool for QoS-aware resource management in Fog computing. First, potential Fog computing applications are grouped in seven CoS according to their QoS requirements. A synthetic database of Fog applications is built from the definition of the intervals that each QoS requirement relevant for a specific Class. At this point, the dataset is pre-processed to convert prior useless data into new data that can be used by ML techniques. Next, a set of popular ML algorithms is selected and puts through the training and testing processes, using the examples in the synthetic database to measure the degree of accuracy and efficiency in their prediction of the CoS to which the application belongs. For this, the synthetic database is contaminated with three different levels of attribute noise. For each noise level, the classifier conducts training and testing to measure the degree of robustness.

This ML-based classification methodology allows the implementation of CoS to manage the traffic in Fog, which constitutes a first step in the definition of QoS provisioning mechanisms in the C2T continuum. Moreover, the integration of IoT, Fog, and Cloud requires efficient management strategies capable of facilitating resource management tasks such as scheduling, allocation, and federation. The classification of Fog applications resolves all these issues; moreover, it is easy to implement into existing devices, and represents a new way of self-adaptive and cognitive network mechanisms [63, 64, 65, 66, 67], which are fundamental for the deployment of autonomic Fog systems in environments with limited availability of processing resources.

For future work, an ML-based classification algorithm will be integrated into the Fog network scheduler, thus enabling the network scheduler to prioritize processing requests. It will also allow more delay sensitive demands to be satisfactorily fulfilled. We also suggest the inclusion of other feature selection methods such as Relief-F, CFS, MCFS, and the Student's t-test in the pre-processing stage for future classification studies in the context of Fog computing. Moreover, we recommend a broad study on the distribution of noise for data analysis oriented to parameters related to Cloud/Fog/edge applications as well as network traffic.

Acknowledgments

This work was supported by the Brazilian National Research Agency CNPq and the Academy of Sciences for the Developing World (TWAS), under process 190172/2014-2 of the CNPq-TWAS program. The authors are also grateful to CAPES (grant No. 88881.145912/2017-01), CNPq (grant No. 307560/2016-3), FAPESP (grants Nos. 2014/12236-1, 2015/24494-8, 2016/50250-1, and 2017/20945-0) and the FAPESP-Microsoft Virtual Institute (grants Nos. 2013/50155-0, 2013/50169-1, and 2014/50715-9) The authors would like to thank reviewers constructive comments.

References

[1] P. Mell, T. Grance, The NIST Definition of Cloud Computing, Tech. Rep. NIST Special Publication (SP) 800-145, National Institute of Standards and Technology (Sep. 2011).

[2] OpenFog Reference Architecture: OpenFog Consortium. Available: https://www.openfogconsortium.org/ra/ [Accessed: 24/05/2017] (2017).

[3] N. Alkassab, C. T. Huang, Y. Chen, B. Y. Choi, S. Song, Benefits and schemes of prefetching from cloud to fog networks, in: 2017 IEEE 6th

International Conference on Cloud Networking (CloudNet), 2017, pp. 1–5.

[4] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, M. Parashar, Mobility-aware application scheduling in fog computing, IEEE Cloud Computing 4 (2) (2017) 26–35.

[5] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, Journal of Network and Computer Applications 98 (2017) 27–42.

[6] A. Kumari, S. Tanwar, S. Tyagi, N. Kumar, R. M. Parizi, K.-K. R. Choo, Fog data analytics: A taxonomy and process model, Journal of Network and Computer Applications 128 (2019) 90–104.

[7] C. C. Byers, Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks, IEEE Communications Magazine 55 (8) (2017) 14–20. doi:10.1109/MCOM.2017.1600885.

[8] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog Computing and Its Role in the Internet of Things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, ACM, New York, NY, USA, 2012, pp. 13–16.

[9] T. Wang, Y. Liang, W. Jia, M. Arif, A. Liu, M. Xie, Coupling resource management based on fog computing in smart city systems, Journal of Network and Computer Applications 135 (2019) 11–19.

[10] R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption, IEEE Internet of Things Journal 3 (6) (2016) 1171–1181.

[11] Y. Shao, C. Li, Z. Fu, L. Jia, Y. Luo, Cost-effective replication management and scheduling in edge computing, Journal of Network and Computer Applications 129 (2019) 46 – 61.

[12] H. R. Arkian, A. Diyanat, A. Pourkhalili, MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications, Journal of Network and Computer Applications 82 (2017) 152–165.

[13] D. M. Batista, N. L. S. d. Fonseca, A survey of self-adaptive grids, IEEE Communications Magazine 48 (7) (2010) 94–100.

[14] D. M. Batista, N. L. S. da Fonseca, Robust Scheduler for Grid Networks Under Uncertainties of Both Application Demands and Resource Availability, Comput. Netw. 55 (1) (2011) 3–19.

[15] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Softw., Pract. Exper. 32 (2) (2002) 135–164. doi:10.1002/spe.432.
URL https://doi.org/10.1002/spe.432

[16] Jin Xu, A. Y. S. Lam, V. O. K. Li, Chemical reaction optimization for task scheduling in grid computing, IEEE Transactions on Parallel and Distributed Systems 22 (10) (2011) 1624–1631. doi:10.1109/TPDS.2011.35.

[17] L. F. Bittencourt, E. R. M. Madeira, N. L. S. D. Fonseca, Scheduling in hybrid clouds, IEEE Communications Magazine 50 (9) (2012) 42–47.

[18] N. L. S. d. Fonseca, R. Boutaba, Cloud Services, Networking, and Management, John Wiley & Sons, 2015.

[19] T. A. L. Genez, L. F. Bittencourt, N. L. S. d. Fonseca, E. R. M. Madeira, Estimation of the Available Bandwidth in Inter-Cloud Links for Task Scheduling in Hybrid Clouds, IEEE Transactions on Cloud Computing 7 (1) (2019) 62–74.

[20] C. Tsai, J. J. P. C. Rodrigues, Metaheuristic scheduling for cloud: A survey, IEEE Systems Journal 8 (1) (2014) 279–291. doi:10.1109/JSYST.2013.2256731.
URL https://doi.org/10.1109/JSYST.2013.2256731

[21] D. Kliazovich, J. E. Pecero, A. Tchernykh, P. Bouvry, S. U. Khan, A. Y. Zomaya, CA-DAG: modeling communication-aware applications for scheduling in cloud computing, J. Grid Comput. 14 (1) (2016) 23–39. doi:10.1007/s10723-015-9337-8.
URL https://doi.org/10.1007/s10723-015-9337-8

[22] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, The Journal of Supercomputing 71 (9) (2015) 3373–3418. doi:10.1007/

s11227-015-1438-4.
URL https://doi.org/10.1007/s11227-015-1438-4

[23] M. A. Ali, A. Esmailpour, N. Nasser, Traffic density based adaptive QoS classes mapping for integrated LTE-WiMAX 5G networks, in: 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–7.

[24] R. Cohen, N. L. S. Fonseca, M. Zukerman, Traffic management and control, Multimedia Communication Network: Technologies and Services, Artech House, Norwood, MA, USA, 1998.

[25] A. Callado, C. Kamienski, G. Szabo, B. P. Gero, J. Kelner, S. Fernandes, D. Sadok, A Survey on Internet Traffic Identification, IEEE Communications Surveys Tutorials 11 (3) (2009) 37–52.

[26] M. Finsterbusch, C. Richter, E. Rocha, J. Muller, K. Hanssgen, A Survey of Payload-Based Traffic Classification Approaches, IEEE Communications Surveys Tutorials 16 (2) (2014) 1135–1156.

[27] S. Zhong, T. M. Khoshgoftaar, N. Seliya, Analyzing software measurement data with clustering techniques, IEEE Intelligent Systems 19 (2) (2004) 20–27.

[28] M. Alhamad, T. Dillon, E. Chang, Conceptual sla framework for cloud computing, in: 4th IEEE International Conference on Digital Ecosystems and Technologies, 2010, pp. 606–610. doi:10.1109/DEST.2010.5610586.

[29] L. Wu, S. K. Garg, R. Buyya, C. Chen, S. Versteeg, Automated sla negotiation framework for cloud computing, in: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013, pp. 235–244. doi:10.1109/CCGrid.2013.64.

[30] V. C. Emeakaroha, I. Brandic, M. Maurer, S. Dustdar, Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments, in: 2010 International Conference on High Performance Computing Simulation, 2010, pp. 48–54. doi:10.1109/HPCS.2010.5547150.

[31] V. C. Emeakaroha, T. C. Ferreto, M. A. S. Netto, I. Brandic, C. A. F. De Rose, Casvid: Application level monitoring for sla violation detection in clouds, in: 2012 IEEE 36th Annual Computer Software and Applications Conference, 2012, pp. 499–508. doi:10.1109/COMPSAC.2012.68.

[32] T. Hobfeld, R. Schatz, M. Varela, C. Timmerer, Challenges of qoe management for cloud applications, IEEE Communications Magazine 50 (4) (2012) 28–36. doi:10.1109/MCOM.2012.6178831.

[33] S. Yang, Iot stream processing and analytics in the fog, IEEE Communications Magazine 55 (8) (2017) 21–27. doi:10.1109/MCOM.2017.1600840.

[34] V. Cardellini, V. Grassi, F. L. Presti, M. Nardelli, On qos-aware scheduling of data stream applications over fog computing infrastructures, in: 2015 IEEE Symposium on Computers and Communication (ISCC), 2015, pp. 271–276. doi:10.1109/ISCC.2015.7405527.

[35] M. Aazam, M. St-Hilaire, C. Lung, I. Lambadaris, Mefore: Qoe based resource estimation at fog to enhance qos in iot, in: 2016 23rd International Conference on Telecommunications (ICT), 2016, pp. 1–5. doi:10.1109/ICT.2016.7500362.

[36] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, K. K. Leung, Dynamic service placement for mobile micro-clouds with predicted future costs, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2017) 1002–1016. doi:10.1109/TPDS.2016.2604814.
URL https://doi.org/10.1109/TPDS.2016.2604814

[37] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, Y. Sun, Qoe-driven joint resource allocation for content delivery in fog computing environment, in: 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1–6. doi:10.1109/ICC.2018.8422843.

[38] R. Mahmud, S. N. Srirama, K. Ramamohanarao, R. Buyya, Quality of experience (qoe)-aware placement of applications in fog computing environments, Journal of Parallel and Distributed Computing 132 (2019) 190 – 203. doi:https://doi.org/10.1016/j.jpdc.2018.03.004.
URL http://www.sciencedirect.com/science/article/pii/S0743731518301771

[39] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), 2017, pp. 89–96. doi:10.1109/ICFEC.2017.12.

[40] J. F. Kurose, K. W. Ross, Computer Networking: A Top-Down Approach (6th Edition), 6th Edition, Pearson, 2012.

[41] M. Böhmer, B. Hecht, J. Schöning, A. Krüger, G. Bauer, Falling Asleep with Angry Birds, Facebook and Kindle: A Large Scale Study on Mobile Application Usage, in: Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services, MobileHCI '11, ACM, New York, NY, USA, 2011, pp. 47–56. doi:10.1145/2037373.2037383.
URL http://doi.acm.org/10.1145/2037373.2037383

[42] S. Khan, S. Parkinson, Y. Qin, Fog computing security: a review of current applications and security solutions, Journal of Cloud Computing 6 (1) (2017) 19. doi:10.1186/s13677-017-0090-3.
URL https://doi.org/10.1186/s13677-017-0090-3

[43] N. A. Ali, A. M. Taha, H. S. Hassanein, Quality of service in 3gpp r12 lte-advanced, IEEE Communications Magazine 51 (8) (2013) 103–109. doi:10.1109/MCOM.2013.6576346.

[44] R. Mahmud, R. Buyya, Fog computing: A taxonomy, survey and future directions, CoRR abs/1611.05539 (2016). arXiv:1611.05539.
URL http://arxiv.org/abs/1611.05539

[45] D. A. Chekired, L. Khoukhi, H. T. Mouftah, Industrial iot data scheduling based on hierarchical fog computing: A key for enabling smart factory, IEEE Trans. Industrial Informatics 14 (10) (2018) 4590–4602. doi:10.1109/TII.2018.2843802.
URL https://doi.org/10.1109/TII.2018.2843802

[46] A. McGregor, M. Hall, P. Lorier, J. Brunskill, Flow Clustering Using Machine Learning Techniques, in: C. Barakat, I. Pratt (Eds.), Passive and Active Network Measurement, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 205–214.

[47] S. García, J. Luengo, F. Herrera, Dealing with Noisy Data, in: Data Preprocessing in Data Mining, Springer International Publishing, Cham, 2015, pp. 107–145.

[48] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, (First Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[49] X. Zhu, X. Wu, Class Noise vs. Attribute Noise: A Quantitative Study, Artificial Intelligence Review 22 (3) (2004) 177–210.

[50] P. Huber, J. Wiley, W. InterScience, Robust statistics, Wiley New York, 1981.

[51] I. E. Naqa, R. Li, M. J. Murphy (Eds.), Machine Learning in Radiation Oncology: Theory and Applications, Springer International Publishing, 2015.

[52] R. Jain, R. K. Jain, Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, edição: 1 Edition, Wiley, New York, 1991.

[53] G. Roffo, Feature Selection Library (MATLAB Toolbox), arXiv:1607.01327 [cs]ArXiv: 1607.01327 (Jul. 2016).

[54] H. Liu, H. Motoda, Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series), Chapman & Hall/CRC, 2007.

[55] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene Selection for Cancer Classification using Support Vector Machines, Machine Learning 46 (1) (2002) 389–422.

[56] D. Cai, C. Zhang, X. He, Unsupervised Feature Selection for Multi-cluster Data, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10, ACM, New York, NY, USA, 2010, pp. 333–342.

[57] K. Pearson, On lines and planes of closest fit to systems of points in space, Philosophical Magazine 2 (11) (1901) 559–572.

[58] Choose Classifier Options - MATLAB & Simulink. Available: https://www.mathworks.com/help/stats/choose-a-classifier.html [Accessed: 21/05/2018] (2018).

[59] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, arXiv:1704.04861 [cs]ArXiv: 1704.04861 (Apr. 2017).

[60] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, arXiv:1801.04381 [cs]ArXiv: 1801.04381 (Jan. 2018).

[61] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, K. Keutzer, SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size, arXiv:1602.07360 [cs]ArXiv: 1602.07360 (Feb. 2016).

[62] R. Kohavi, A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 1137–1143.

[63] D. M. Batista, N. L. S. da Fonseca, F. K. Miyazawa, F. Granelli, Self-adjustment of resource allocation for grid applications, Computer Networks 52 (9) (2008) 1762–1781.

[64] A. Mesodiakaki, F. Adelantado, L. Alonso, C. V. Verikoukis, Energy-efficient user association in cognitive heterogeneous networks, IEEE Communications Magazine 52 (7) (2014) 22–29. `doi:10.1109/MCOM.2014.6852079`.
URL `https://doi.org/10.1109/MCOM.2014.6852079`

[65] D. M. Batista, N. L. S. d. Fonseca, F. Granelli, D. Kliazovich, Self-Adjusting Grid Networks, in: 2007 IEEE International Conference on Communications, 2007, pp. 1–5.

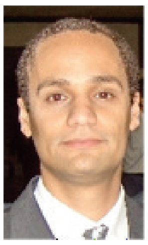
[66] G. Baggio, R. Bassoli, F. Granelli, Cognitive software-defined networking using fuzzy cognitive maps, IEEE Trans. Cogn. Comm. & Network-

ing 5 (3) (2019) 517–539. doi:10.1109/TCCN.2019.2920593.
URL https://doi.org/10.1109/TCCN.2019.2920593

[67] K. Merchant, S. Revay, G. Stantchev, B. Nousain, Deep learning for rf device fingerprinting in cognitive communication networks, IEEE Journal of Selected Topics in Signal Processing 12 (1) (2018) 160–167. doi:10.1109/JSTSP.2018.2796446.

Author biographies

Judy C. Guevara received her degree in control engineering (2009) and M.Sc. degree in Information and Communication Sciences (2012) from the Universidad Distrital Francisco José de Caldas, Bogotá, Colombia, and is currently working toward the Ph.D. degree at the Institute of Computing, State University of Campinas (Unicamp), Brazil. Her awards include the Young Researchers and Innovators "Virginia Gutiérrez de Pineda"fellowship (2010), supported by the Colombian Administrative Department of Science, Technology and Innovation, COLCIENCIAS; and the CNPq-TWAS Postgraduate Fellowship (2014). Her research interests focus on resource management and scheduling in cloud and fog computing networks.



Ricardo da S. Torres received the B.Sc. degree in computer engineering and the Ph.D. degree in computer science from the University of Campinas (Unicamp), Brazil, in 2000 and 2004, respectively. He was the Director of the Institute of Computing, Unicamp, in 2013, where he is currently a Full Professor of Computer Science. He is the Cofounder and a member of the RECOD Laboratory, where he has been developing multidisciplinary research involving image analysis, content-based image retrieval, databases, digital libraries, and geographic information systems. He has authored or coauthored over 100 articles in refereed journal and conferences and serves as a PC member for several international and national conferences.

Nelson L. S. da Fonseca received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1994. He is currently a Full Professor with the Institute of Computing, State University of Campinas, Campinas, Brazil. He has authored or coauthored over 400 papers and has supervised over 60 graduate students. Prof. Fonseca is currently the Vice President

Technical and Educational Activities of the IEEE Communications Society (ComSoc). He served as the ComSoc Vice President Publications, Vice President Member Relations, Director of Conference Development, Director of Latin America Region, and Director of On-Line Services. He is the Past Editor-in-Chief of IEEE Communications Surveys and Tutorials. He is Senior Editor of the IEEE Communications Magazine, an Editorial Board Member of Computer Networks, Peer-to-Peer Networking and Applications. He was a recipient of the 2012 IEEE Communications Society (ComSoc) Joseph LoCicero Award for Exemplary Service to Publications, the Medal of the Chancellor of the University of Pisa, in 2007, and the Elsevier Computer Network Journal Editor of Year 2001 Award.

*Author Contributions Section

Universidade Estadual de Campinas
Instituto de Computação

Authors´Contribution to the manuscript
“On the Classification of Fog computing applications: A Machine Learning Perspective”

Judy Guevara – the manuscript is part of her Ph.D. thesis. She programmed, generated the data , co-wrote and revised the manuscript.

Ricardo Torres- advised on machine learning techniques, co-wrote and revised the manuscript.

Ricardo Torres- thesis supervisor, advised on fog and networking content, co-wrote and revised the manuscript.

[illegible]

Prof Nelson Luis Saldanha da Fonsca, Judy Guevara and Ricardo Torres
Institute of Computing, State University of Campina, Brazil

***Conflict of Interest**

The authors declare that there is no Conflict of Interest in the submission of the manuscript “On the Classi_cation of Fog computing applications: A Machine Learning Perspective”, Judy C. Guevara, Ricardo da S. Torres, Nelson L. S. da Fonseca

[illegible]