# Applying events stream processing to network Online Failure Prediction

| | |
|---|---|
| Journal: | *IEEE Communications Magazine* |
| Manuscript ID | COMMAG-16-01135.R2 |
| Topic or Series: | Network & Service Management Series |
| Date Submitted by the Author: | n/a |
| Complete List of Authors: | Dueñas, Juan; Universidad Politécnica de Madrid, Telemática, ETSI Telecomunicación<br>Navarro, José; Universidad Politécnica de Madrid, Telemática, ETSI Telecomunicación<br>Parada, Hugo; Universidad Politecnica de Madrid, Electrónica y Telemática, ETSIS de Telecomunicación<br>Andión, Javier; Universidad Politécnica de Madrid, Telemática, ETSI Telecomunicación<br>Cuadrado, Félix; Queen Mary University of London, School of Electronic Engineering and Computer Science |
| Key Words: | online failure prediction, events stream processing, training and prediction models, proactive fault management, Spark streaming |
| | |

# Applying event stream processing to network Online Failure Prediction

Juan C. Dueñas[1], José M. Navarro[1], Hugo A. Parada G.[1], Javier Andión[1], Félix Cuadrado[2]
[1]{juancarlos.duenas, josemanuel.navarro, hugoalexer.parada, j.andion}@upm.es
[1]Universidad Politécnica de Madrid, Spain
felix.cuadrado@qmul.ac.uk
[2]Queen Mary University of London, UK

## Abstract

Predicting failures on networks and systems is critical in order to maintain high uptime rates. Online Failure Prediction (OFP) techniques use machine learning and predictive analytics to generate failure models that can be applied to computer network data. These techniques can be provisioned on state of the art stream processing systems, such as Spark Streaming, in order to cope with the scalability challenges from the base data. A big challenge with OFP is selecting the right information to process, as well as the appropriate features in order to achieve high accuracy in predicting failures on complex, interconnected systems. In this paper we describe an Online Failure Prediction system built over Apache Spark that takes a repository of network management events, trains a Random Forest model and uses this model to predict the appearance of future events in near real time. We show through our experiments the usefulness of network management events for accurate predictions, and the advantages of the proposed system in terms of predictive quality, cost and ease of deployment.

## 1. Online Failure Prediction in practice

Effective management of large-scale distributed systems requires a proactive approach to failure management. Current research efforts propose self diagnosing systems [1--3] that forecast their future state in order to predict failures, thus reducing the time to repair. Online Failure Prediction (OFP) [4] aims to provide accurate predictions with minimal intrusiveness, observing the current state of the system and applying prediction rules to foresee future states. OFP techniques frequently define their prediction rules based on the experience of domain experts and system administrators.

Recently, Machine Learning prediction methods have been applied to Network Management systems [5]. Unlike traditional approaches, these algorithms achieve high accuracy by automatically detecting rules and patterns from the studied system. Ensemble methods, such as Random Forest [6], have become one of most promising machine learning approaches for prediction. These techniques train an ensemble of hundreds of simple models, combining their individual results to capture feature interactions, providing highly accurate predictions. Random Forest methods have been successfully applied to several network management problems, such as hardware failure prediction [7], and network

intrusion detection [8].

However, ensemble methods are challenging to implement in online scenarios with strict time prediction requirements; these techniques require training and executing a large number of models, which can be challenging to accomplish within strict temporal prediction deadlines. On the other hand, state-of-the-art distributed stream processing platforms such as Apache Storm and Spark Streaming [9], are a natural candidate to enable timely accurate predictions over a large volume of event data.

This paper presents an online failure prediction system for network events based on ensemble methods. The system performs live predictions with high scalability, exploiting Apache Spark to parallelize model training and prediction tasks. We validated the system with an industrial dataset of network management events from a major Spanish bank. We also present the main requirements and design decisions of the system, as we believe they can be applicable to other online network management scenarios.

# 2. The Failure Prediction System

The motivation of our work was to aid a Spanish bank to design a proactive management system enabling system administrators to react to critical events before they appear on this network. We worked with a dataset of network management events occurring at the bank datacenter between September 2014 and June 2015. The managed network is composed by 22 devices: one router, seven switches and fourteen virtual machines. The dataset contains 21442 event traces, divided into 84 different types of events classified into four different severity categories: Critical (4 types of events), Major (11), Minor (7) and No Severity (64). These events range from CPU overload on a host to trivial network status messages.

Our goal was to predict the appearance of events of a given type in a device, so the number of types of events of interest has an upper bound of 1848 (84 different types times 22 devices), but the actual number is 575 - lower than threshold as not all types of events happen in all devices. The dataset shows a very sparse rate of events arrival -the busiest 5 minutes period accounted for 503 events.

We first performed preliminary time series analysis of the recorded events, revealing no periodicity in the data. Hereafter, we explored Machine Learning prediction techniques such as Random Forests. In order to apply this technique, we first had to train a model from the historic event dataset that captures the behaviour of the managed system. This first phase is called training. The created model - a forest of decision trees - will be used to predict future failures from the live feed of management events.

As predictions were to be executed online, we set sliding time windows [4] for both past events to consider as well as future predictions. The system uses the trained model to predict what might happen on a prediction window, based on past occurrences from the observation window. Observation windows with the recent events are generated periodically

and sent as a stream of events to the prediction element. Some of the design parameters are:

- ***time unit:*** discrete time interval in which an event can be observed. Burstiness analysis from our dataset revealed that one second was accurate enough for our dataset. Multiple events of the same type happening in the same time unit are indistinguishable from a single occurrence.
- the ***observation window*** is the period of time the prediction system considers past events in order to perform a prediction. In our case, previous analysis of predictive power with different window lengths led us to use a 5-minute observation window.
- the ***prediction window*** is the period of time in the future for which a prediction is made. A 5-minute window length was defined as it was deemed to be a reasonable trade-off between prediction accuracy and notice time to address the incoming incident by system administrators before it actually happens. Note the prediction function does not aim to specify the exact time within the window when the event will happen if predicted.
- ***model features:*** the occurrence or not of a type of event in an element of the managed system. Management systems usually have a large number of event types - 575 for ours; but usually only a fraction of them require reaction, those marked as Critical and Major. Hence, we only predict the critical and major events that may happen in each network node, although all types are considered to create the predictors.

We designed a system providing these main functions: 1) to train Random Forest predictive models based on the historical dataset of management events, 2) to transfer the models to the online prediction part, and 3) to predict critical and major events in a time-ahead window based on the observation window events stream, while trying to incur in minimal latency. We describe the system architecture in the following subsection.

## 2.1. System Architecture

The system architecture is composed by two main subsystems, as depicted in Fig 1. The training subsystem (upper side) performs offline computations from past event data collected from the management system. The prediction subsystem (lower side) performs online failure prediction based on the events captured by the management system and the trained decision model.

The input to our system is the stream of events obtained by the management system, rather than the metrics on network and resource usage collected from the managed system. This way, we isolate the prediction system from the managed network, as only a direct communication socket between the predictor and the management system is required.

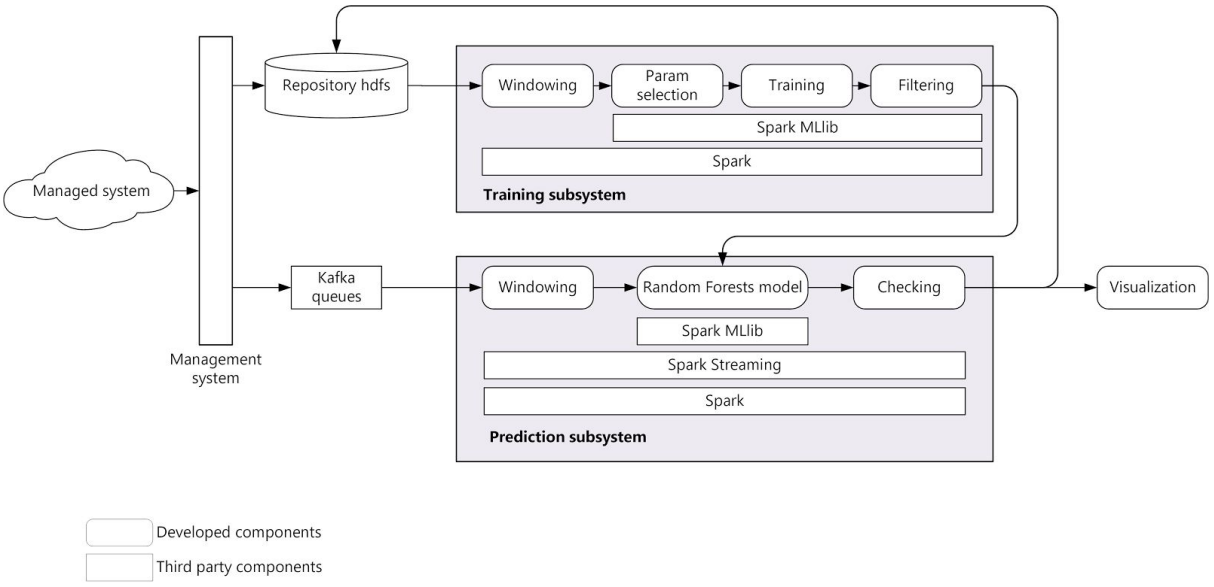


*Fig. 1: OFP system Architecture*

We chose the “data flow” architectural pattern to develop the prediction system, as it nicely fits to the transformation of streams of events that we get as input. Data flow architecture defines the data ingestion and the sequence of transformations to be performed up to the final result. The event streams generated by the management system are aggregated and sliced into observation windows; the prediction subsystem predicts future events in the lookahead window based on this input.

We have chosen Apache Spark 1.6.0[1] as the base processing platform for the failure prediction system. Spark provides a dataflow programming model with in-memory computations, automatically distributing Resilient Distributed Datasets [9], and processing them in parallel across the computation cluster. Spark also supports the micro batch stream processing model through Spark Streaming [10], where events from a stream are automatically aggregated in computation windows, and processed generating an output stream. While the micro batch model introduces a minimum delay (about 1 second), this time resolution is appropriate to perform event ingestion for network and system management purposes, in particular for prediction [5]. The processing platform can distribute each single transformation to a different processor or core, allowing the ingestion of a large number of events and the execution of many predictions at once if required.

We built on top of Spark MLLib for our Random Forest implementation. This library has been shown to be the most appropriate option to implement stream-based machine learning systems, over other similar systems such as Apache Storm, or Spark ML [11--13]. We must note though that, while Spark abstracts from many complexities of writing parallel programs, implementing complex processing flows that process large amounts of data require significant tuning and experimentation [14]. We will share our experience building the system in the following subsections.

[1] http://spark.apache.org, last accessed June 2017

## 2.2. The training subsystem

The upper part of Fig. 1 shows the training subsystem, which takes as input the historic dataset of events in order to train the Random Forest ensemble model. Input data is organized in text lines that show the occurrence of an event in a node of the managed network, providing the event type and the time it happened. The management system transfers this information in batch mode to the HDFS[2] (Hadoop Distributed File System, able to handle large files). The output of the training subsystem is an in-memory Random Forest prediction model, to be used by the prediction subsystem.

We illustrate an example execution of the training subsystem in the left side of Fig 2, where a model is built for a dataset containing 4 different event types (for clarity we consider each event happens in a different node). The HDFS input is an ordered stream of events, including timestamp information for each event. The windowing process then creates one five minute observation window every second, for the whole dataset timespan. We create a Random Forest model for each event we aim to predict (critical and major events on each network node). The input to train the RF model is the collection of observation windows (organized as vectors of observed events in each window), annotated with information about whether the event of interest happened within the prediction window (one 1-element vector for each event type for each window). For the purposes of training we consider the prediction window starts right after the end of observation window without delays, and lasts five minutes. There is one model generator for each event type, taking as input the complete set observation windows with event appearance annotations. Each model generator will create the Random Forest model for its type of event.

We tuned the hyperparameters of the RF generation algorithm the following way: we fixed the number of trees to 50 per predicted event type following the suggestions made on [15]; we set the maximum number of split layers each decision tree can have (maximal depth) to 9, close to the binary log of number of the whole set of types of events ($\log_2(575) \sim 9.16$). We used the Spark MLLib RF training algorithm in our subsystem.

[2] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, last accessed June 2017

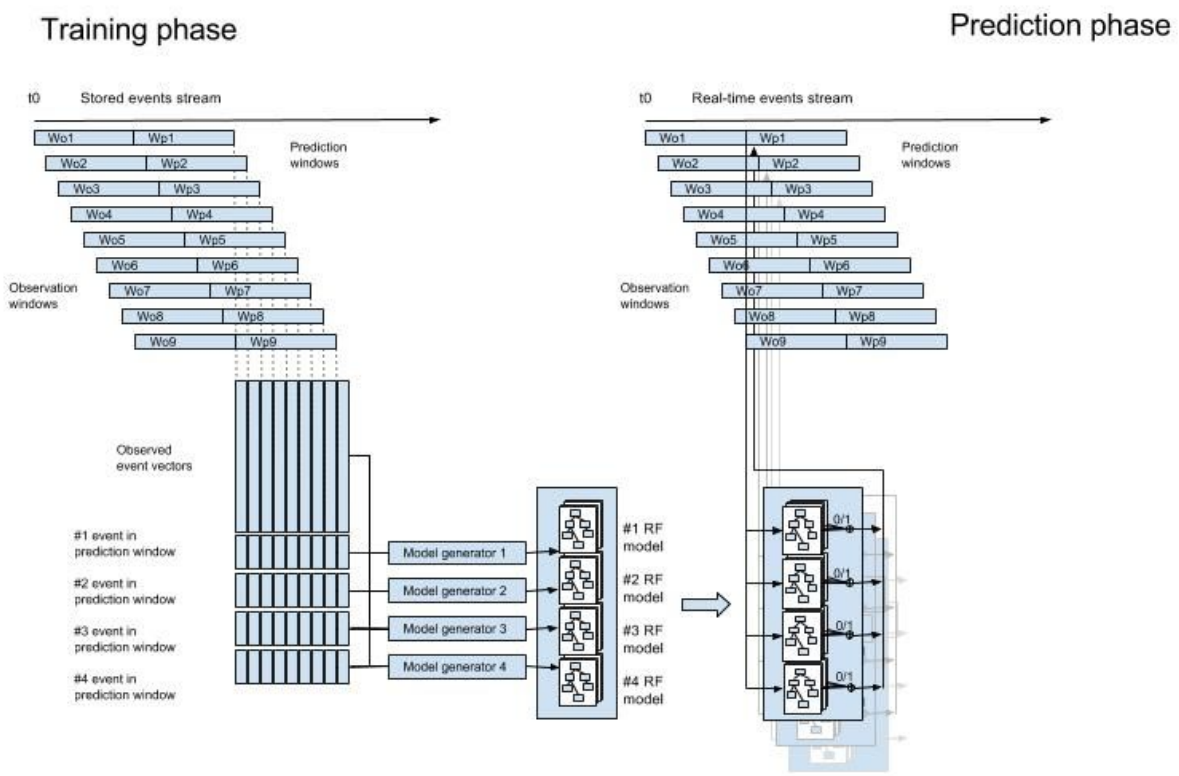


*Fig. 2: Example execution of the training subsystem(left) and prediction subsystem (right)*

After configuring these hyperparameters of the RF model, we trained the system for the whole array of events in each node. A subset of these models was later selected in the "Filtering" activity in the training subsystem. As our system is devised as a production software, we imposed a very strict criterion to accept a model: only the events whose models got an F1 score [4] higher than 0.75 were accepted for evaluation (F1 score is a value ranging from 0 to 1, where 1 would be a perfect model, built by taking the false negative, true positive and false positive cases of a predictor). Regarding evaluation schemes, we used cross validation, a standard technique for Machine Learning models performance evaluation; it ensures that every part of the dataset will be part of both the training and testing phase, something that other evaluation metrics such as the Out of Bag process does not provide. We applied 10-fold cross validation, in order to reduce the variance as much as possible without excessively increasing computation time. This final set of Random Forest models is transferred to the prediction subsystem. In our dataset, out of 84 event types potentially appearing on 22 network nodes (a maximum of 84*22=1848 different event types), only 108 event types occurred with enough frequency for their Random Forest models to meet the F1 score requirement of 0.75.

Training a set of RF models is a complex, and computationally expensive task, although we take advantage of RDD transformation parallelization to speed up its completion. Trained models will keep similar prediction accuracy, as long as the event source does not experience significant drift, and the model was trained on an information rich dataset

(number of samples, temporal extension, number of appearances of each event type and relationships between event types).

## 2.3. The prediction subsystem

The prediction subsystem will load the model trained in the training subsystem, in order to perform predictions on the stream of management events (lower part of Figure 1). It must be noted that the prediction phase operates online, in contrast with the offline training phase. The stream of events is fed to the prediction subsystem through Kafka[3] queues.

The prediction flow is shown on the right side of Fig. 2. Each time an observation window opens, and for the time it lasts, Spark Streaming sets a data chunk with the events happening during that time span (windowing process). We decided to produce a new prediction each second for each of the 108 events of interest - as a compromise between prediction granularity, machine load and capacity to produce understandable results. Each one of the 108 RF-based event prediction models runs as an independent task that predicts the appearance of one type of event. Each model is a Random Forest of 50 trees, with majority voting determining whether the event will happen (or not) at any point during the prediction window. All the 108 predictions must be made within the one second window, otherwise the online system becomes saturated. Each prediction window starts right after the end of its observation window.

This way, each second, 108 positive or negative predictions are obtained about the appearance of each event over the next 5 minutes (prediction window), based on the previous 5 minutes (observation window); the combination of window and prediction intervals means that, for each event of interest in a given second, 300 predictions are open. Interpreting the semantics of multiple predictions for the same event at once can be confusing for human operators, increasing the need for adequate visual representations. Because of that, we designed a visualisation component in order to provide system administrators with a global overview on both past state and future predictions. The aim of this visual component is to guide system administrators on corrective actions such as killing a process, stopping a node or launching a new Web server instance.

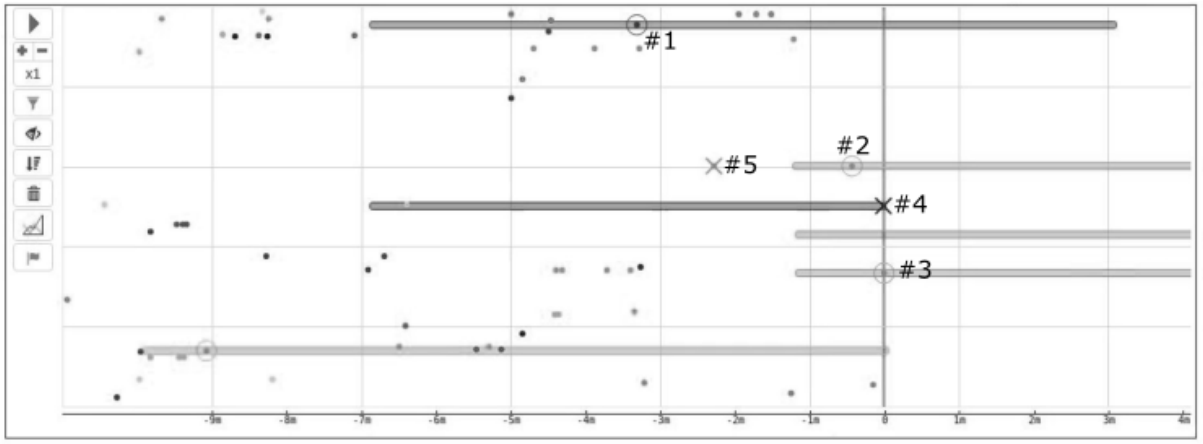


*Fig. 3: Visualization example*

[3] http://kafka.apache.org, last accessed September 2017

Figure 3 shows part of the visualisation prototype we have developed to help in the interpretation of results. The x axis shows time, with the vertical line showing the current time. Different events (type and node) appear at different heights, with horizontal bars representing that the event is predicted to occur over the segment time range. Points represent the occurrence of non-interesting events. Let us remark that, for a positive prediction (the event is expected to happen), no indication is given about the specific time of the potential appearance of the predicted event; instead, the event would appear at any time of the prediction window if the forecast is positive; it might even appear more than once. Hence, segments on the right hand side of the vertical line represent current positive predictions of events, whereas past events and predictions appear on the left of the timeline. We visualise the outcome of our predictions with the following representation:

- The event was predicted to happen and it does - for the remaining prediction window this is marked as True Positive - it is signalled with a small circle over the bar that marks the prediction (on times: -3.2 minutes in top event timeline #1, -0.4 minutes on the second timeline #2, 0 minutes on the fifth one #3).
- The event was predicted to happen and it does not - this can only be known at the end of the prediction window and it renders a False Positive. This appears as a prediction window that finishes without the actual appearance of the event; it is marked with a cross (see the end of the third event timeline from the top #4).
- The event was not predicted to happen but it does happen. Whenever a critical or major event appears without a prediction bar, this can be labelled as a False Negative (the isolated cross at -2 minutes #5).
- The event was not predicted to happen and it does not - which can only be ensured at the end of the prediction window and renders a True Negative. Nothing is shown so visualization is kept simple.

We deployed the complete system[4], both training and prediction subsystems, in two different settings: a laptop equipped with an Intel i7-3720QM 2.6GHz processor, 8 virtual cores with HyperThreading, 16GB of RAM running Ubuntu x64 15.10 as operating system; and a hardware cluster composed by the master machine and seven workers - each node is a HP ProLiant SL210t G8 with 2 Intel Xeon processors E5-2630v2 (2.6 GHz), 6 cores each, for 12 physical cores (up to 24 logical cores with HyperThreading); 32 GB of RAM and 3TB SATA storage.

We measured the memory requirements of the trained models, and found an upper limit of 4.4 MB per model, rendering the total memory size for a production system at most at 108 * 4.4 = 475.2 MB. Although additional memory is required for the observation and prediction windows, this figure is fairly low.

The prediction process inserts a delay between the end of the observation window and the start of the prediction window; this delay cannot be higher than the prediction update period. Otherwise, saturation would happen. In our experiments on both deployment settings we found that predicting the occurrence of all 108 events of interest took around 350

[4] The code is available at https://github.com/jandion/SparkOFP, accessed Sep 2017.

milliseconds in the portable computer, and 330 in the cluster. Each prediction requires traversing all trees in the model, rounding up to treesPerModel * number of models = 50 * 108 = 5400 decision trees. Trees have a variable number of depth levels, as 9 is just each tree's depth upper bound. This suggests that Spark's scheduling algorithms are working far from the saturation level. Both deployments show an adequate latency and capacity for the problem at hand, and seem to provide a valid basis for more demanding environments. Performance of the prediction phase depends on the number of event types of interest, the length of the prediction window and the time pace new prediction windows are opened, more than on the actual number of events in each observation window.

We tested the prototype in a practical setting, over a validation period of 3 months. The visualisation functionality received very positive feedback from system administrators, as they detected numerous critical or major events prior to appearance, having additional time to solve the problem following their in-house practices. The quality of the predictions was high, as we expected after the rigorous filtering process performed when selecting predictors; F1 score values for the 108 event types comprising critical and major events varied between 0.75 and 0.98.

# 3. Conclusions

In summary, we have shown how to build a proactive fault management system based on Random Forest. Both the training and the prediction processes are supported by the same architecture, analyzing the events emitted by the management system. Configuration parameters have been carefully chosen to optimize the quality of the predictors while keeping the computation time at a minimum. The usage of an online processing engine such as Spark, enhanced with the Spark Streaming module and the implementation of Random Forest algorithm in Spark MLLib, proved suitable to build such a prediction system.

Our future plans are to test and compare additional predictive techniques or ensembles of these. We are particularly interested in techniques capable of incremental training, as they would be suitable even under changes to the network topology. This approach would also require the application of different architectural patterns; working with different datasets to identify the conditions for application, and adding diagnosis capabilities.

# Bibliography

[1] Guan, Qiang, and Song Fu. "auto-AID: A data mining framework for autonomic anomaly identification in networked computer systems." *International Performance Computing and Communications Conference*. IEEE, 2010.

[2] Zaman, Faisal, et al. "A recommender system architecture for predictive telecom network management." *IEEE Communications Magazine* 53.1 (2015): 286-293.

[3] Chaparadza, Ranganai. "UniFAFF: a unified framework for implementing autonomic fault management and failure detection for self-managing networks." *International Journal of Network Management* 19.4 (2009): 271-290.

[4] Salfner, Felix, Maren Lenk, and Miroslaw Malek. "A survey of online failure prediction methods." *ACM Computing Surveys (CSUR)* 42.3 (2010): 10.

[5] Kalegele, Khamisi, et al. "Four Decades of Data Mining in Network and Systems Management." *IEEE Transactions on Knowledge and Data Engineering* 27.10 (2015): 2700-2716.
[6] Rokach, Lior. "Decision forest: Twenty years of research." *Information Fusion*27 (2016): 111-125.
[7] Pitakrat, Teerat, André van Hoorn, and Lars Grunske. "A comparison of machine learning algorithms for proactive hard disk drive failure detection." *Proceedings of the 4th international ACM Sigsoft symposium on Architecting critical systems*. ACM, 2013.
[8] Buczak, A. L., & Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), 1153-1176.
[9] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.
[10] Zaharia, Matei, et al. "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters." *HotCloud*, 2012, vol. 12, p. 10-16.
[11] Zheng, Jiang, and Aldo Dagnino. "An initial study of predictive machine learning analytics on large volumes of historical data for power system applications." In *IEEE International Conference on Big Data (Big Data), 2014*. IEEE, 2014.
[12] Richter, Aaron N., et al. "A Multi-Dimensional Comparison of Toolkits for Machine Learning with Big Data." In *IEEE International Conference on Information Reuse and Integration (IRI), 2015*. IEEE, 2015.
[13] Sun, Ke, et al. "An Improvement to Feature Selection of Random Forests on Spark." In *IEEE 17th International Conference on Computational Science and Engineering (CSE), 2014*. IEEE, 2014.
[14] Landset, Sara, et al. "A survey of open source tools for machine learning with big data in the Hadoop ecosystem." *Journal of Big Data* 2.1 (2015): 1.
[15] Oshiro, Thais Mayumi, Pedro Santoro Perez, and José Augusto Baranauskas. "How many trees in a random forest?." *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer Berlin Heidelberg, 2012.

More information about the dataset and the model training process can be found on several annexes at: https://goo.gl/kThRgD.

## Acknowledgments

The authors would like to express their gratitude to Produban, who inspired and motivated this challenge as a real business case and provided all necessary assistance to carry out this work. The work performed by José M. Navarro has been funded by Ministerio de Educación de España under grant BFPU-2014-03209.

# Biographies

**Juan C. Dueñas [SM] (juancarlos.duenas@upm.es)** obtained a Degree on Telecommunications Engineering by Universidad Politécnica de Madrid, Spain, 1991, and a PhD by UPM in 1994. He is currently Deputy Vice-President for Research in Universidad Politécnica de Madrid.

**José M. Navarro (josemanuel.navarro@upm.es)** obtained an MSc on Telecommunications Engineering by Universidad Miguel Hernández de Elche, Spain on 2013. He is a PhD candidate at Escuela Técnica Superior de Ingenieros de

Telecomunicación in UPM, Madrid, Spain. His research interests are distributed systems management through applied machine learning and the Internet of Things.

**Hugo A. Parada G. (hugo.parada@upm.es)** is an Assistant Professor in Universidad Politécnica de Madrid (UPM) at Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación. His current areas of research include machine learning, big data, and cloud computing. He received his Ph.D. in Telecommunications engineering from UPM in 2010.

**Javier Andión [M] (j.andion@upm.es)** is a Telecommunications Engineer by Universidad Politécnica de Madrid, Spain. He is pursuing a PhD on proactive failure prediction with Machine Learning at Escuela Técnica Superior de Ingenieros de Telecomunicación in UPM, Madrid, Spain. He has been involved in the organization of several workshops and congress related with artificial intelligence, robotics and programming challenges.

**Félix Cuadrado [M] (felix.cuadrado@qmul.ac.uk)** is Senior Lecturer (Associate Professor) in the School of Electronic Engineering and Computer Science of Queen Mary University of London. He obtained a MEng in Telecommunications Engineering from UPM in 2005, and a PhD from UPM in 2009. His research interests include large-scale data processing systems, and Internet-scale applications.