Executive Summary of the Thesis

Anomaly detection as-a-Service for Predictive Maintenance

Laurea Magistrale in Computer Science And Engineering - Ingegneria Informatica

Author: Daniele De Dominicis

Advisor: Prof. Manuel Roveri

Co-advisors: Dott. Alessandro Falcetta, Ing. Mirco Zanetti

Academic year: 2021-2022

1. Introduction

Cloud Computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. In particular, in the recent years Machine Learning as-a-Service (MLaaS) is playing a fundamental role in the Cloud world. The support of MLaaS approach resides in the ability to provide users and companies, via web-console or API, with ready-to-use or remotely trainable Machine Learning (ML) models leveraging the high performance computing and memory abilities of Cloud Computing systems. This thesis aims at build an innovative client-server architecture that allows to offer Anomaly detection as-a-Service via RESTful APIs, based on supervised and unsupervised ML algorithms, with three different types of services: the training of a ML model, the update of a ML model and the inference of a sample data through a ML model. In the literature of MLaaS most of the efforts focused on images, video and textual information, while very little attention has been posed to sensor data sampling in industrial environments. From this perspective, the design and development of Cloud-based MLaaS tool for the identification of anomalies in the industrial environment is a novel and promising research area. Moreover, a real case study is treated in which Eisenmann Italia S.r.l. company wants to build a monitoring system for the identification of anomalies in the perspective of predictive maintenance in Industry 4.0. And a client-server architecture for Anomaly detection as-a-Service is a solution tailored to the business's needs.

2. Related Literature

In the literature of MLaaS the applications are far from Anomaly detection. Tay [2] applied deep learning algorithms within the area of wood identification, to create a MLaaS product. Their tool consists of individual cloud-based services for data collection, data verification, model training, internal testing, and inference. Instead, to tackle the cost of bug fixing within software engineering, modern debugging and testing methods moved from finding to predicting the bugs [3]. The field of bug fixing is another area that leverages ML predictions. By proposing bug prediction via MLaaS the authors provide Bug Prediction as-a-Service (BPaaS). MLaaS can be leveraged in other areas as well and of the challenges within MLaaS are attacks on network data. Rajagopal et al. [4] propose a solution within network intrusion detection applying MLaaS and by that increasing time-efficiency. The goal is to correctly classify an attack as such.

3. Background

Anomalies can appear in different forms. Commonly, three different types of anomalies exist: **Point anomalies** in which a point deviates significantly from the rest of the data. **Collective anomalies** in which a collection of related data instances is anomalous with respect to the entire data set. **Contextual anomalies** are observations or sequences which deviate from the expected patterns within the time series, however, if taken in isolation they may be within the range of values expected for that signal. Anomaly detection methods can be divided in three main categories: statistical methods, Machine Learning methods and Deep Learning methods. **Statistical** methods utilize historical data to model the expected behavior of a system. When a new observation is received it is compared against the current model for that system and if it does not fit within that model it is registered as an anomaly. **Machine Learning (ML)** is the study of computer algorithms which are able to improve automatically in solving a certain task. ML algorithms use a mathematical model which could be optimized using sample data (training data) or not. There are at least three different paradigms used to build the model, based on the availability of the labels in the data: **Supervised anomaly detection** techniques assume the availability of a training data set which has labeled instances for normal as well as anomaly class. **Semi-Supervised anomaly detection** techniques assume that the training data has labeled instances for only the normal class. **Unsupervised anomaly detection** techniques do not require label training data and thus are most widely applicable.

4. Architecture

The work aims to build a client-server architecture that allows to offer Anomaly detection as-a-Service via RESTful APIs. The Anomaly detection as-a-Service solution allows the client to make requests to a server, in order to train ML models for the identification of anomalies from the data sent by the client and to make inference on a single data from the previously trained models. Since the identification of anomalies can refer to a wide range of applications and consequently data, in this work the focus is on data collected by accelerometers installed on industrial machines, without any loss of generality. Accelerometers will sample at a certain sampling rate and the data will be saved within Csv files.

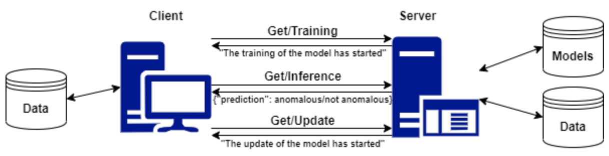
Client
Server
Get/Training
"The training of the model has started"
Get/Inference
{"prediction": anomalous/not anomalous}
Get/Update
"The update of the model has started"
Data
Models
Data

Figure 1

The client-server architecture for Anomaly detection as-a-Service is shown in Figure 1 and it is designed to meet the following client requirements: **Training** of a model, **Inference** of a data sample and **Update** of a model. Client side, the Python scripts have been developed in order to make easy for the client the use of the services. Scripts allow to take a single data or a series of data as input and serialize them in JSON format, to send the request and the JSON (in its body) at the appropriate server API. Server side it is necessary to implement the APIs defined by appropriate URLs. They must allow the client to reach the server and send the above mentioned requests along with the data in JSON format. After defining the APIs, the server is ready to receive the client requests and execute the appropriate "service" code. The three services are analyzed in detail for both client and server. The figure 2 shows the Training service pipeline.

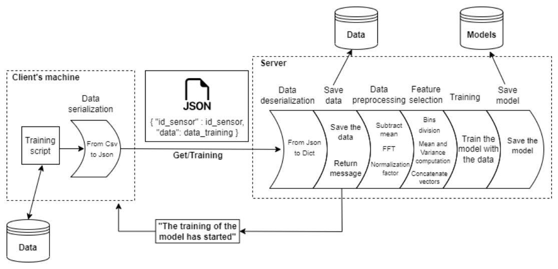

Figure 2

The client-side Python script: takes as input the path of a folder containing the Csv files related to the data samples of a sensor, in order to carry out the training of the model. Then it transforms the Csv files into a single JSON with the structure displayed in the Figure 2, where id_sensor is the identification of a sensor. Finally it makes a GET request containing the JSON to the appropriate API of the server. The server listening: receives the GET request from the client with the JSON in the body of the request. It saves the JSON in the DB Data with the appropriate nomenclature, that is the id_sensor (id_sensor.json) and sends the answer to the client notifying the start of the training of the model ending the communication with client. Then it performs the preprocess of the data first and then the selection of the features. Then it performs the training of the ML model using the modified data. Finally it saves the model in the DB Models with the appropriate nomenclature, that is the id_sensor (id_sensor.joblib). The figure 3 shows the Inference service pipeline.

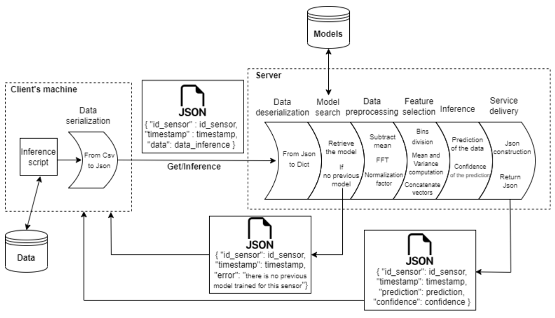

Figure 3

The client-side Python script: takes as input the Csv file related to the data sample of a sensor, in order to carry out the inference of the sample. Then it transforms the Csv file into a JSON with the structure displayed in the Figure 3. Finally it makes a GET request containing the JSON to the appropriate API of the server. After receiving the reply from the server containing the JSON, it saves the inference result in the folder "results" (if not present is created) as id_sensor.timestamp.json. The server listening: receives the GET request from the client with the JSON in the body of the request and extracts the model inside the DB Models (if present) through the id_sensor, otherwise the server returns the JSON containing the error and it ends the requested service. Then it performs the preprocess of the data first and then the selection of the features. It uses the model to inference the modified data. Then it sends the response to the client containing the JSON with the structure shown in the Figure 3. In particular, the "prediction" key contains the value 1 if the model has evaluated the sample as nominal, otherwise it contains the value -1. Instead, Confidence expresses the ability of the model to provide a given prediction and it is a value between 0 and 1; it is an additional semantics to the single 1/-1 of the prediction, since it makes sense to look at how the measure evolves over time. The figure 4 shows the Update service pipeline.

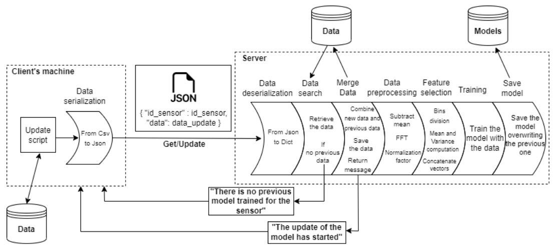

Figure 4

The client-side python script: takes as input the path of a folder containing the Csv files related to the data samples of a sensor, in order to update the sensor model with new samples.Then it transforms the Csv files into a single JSON with the structure displayed in the Figure 4. Finally it makes a GET request containing the JSON to the appropriate API of the server. The server listening: receives the GET request from the client with the JSON in the body of the request and extracts the JSON containing the previous sensor data from the DB Data through the id_sensor (if present), otherwise the server returns the error message and it ends the requested service. Then it joins the JSON containing the previous sensor data with the JSON sent by the client, and saves it in the DB Data overwriting the previous JSON. It sends the answer to the client notifying the start of the training of the model with the integration of the new data and ends the communication with the client. Then it performs the preprocess of the data contained in the new built JSON first and then the selection of the features. Finally it performs model training using the modified data and saves the model in the DB Models overwriting the previous model(Figure 4).

5. Operational Steps

As noted above, the work focuses on data collected by accelerometers installed on machinery. This means that the data we deal with are signals in time domain. However signals in time domain do not accurately characterize the vibrations captured by the accelerometers. Therefore the signal has to be transformed in the signal in frequency domain. In particular it was used the Fast Fourier Transform (FFT) algorithm to compute the discrete Fourier Transform and to convert a signal from time to frequency domain. Basically, if the signals are sampled at a rate f_s, then the FFT will return the frequency spectrum up to a frequency of $f_s/2$. Applying the FFT on the signals of interest, one can see particular peaks at 0 Hz. Since the data are extracted from accelerometers, it indicates the presence of a static acceleration field influencing the sensors (gravity). This kind of bias, especially for an accelerometer, is not desired. One way to fix this is to subtract the mean of the signal to the signal itself. Thus the first operation is to subtract the average value from the signal in the time domain in order to remove the gravity acceleration. Then apply the FFT to the signal and take from the output of the FFT function the absolute value of the first $N/2$ points of the signal in the frequency domain. Finally the signal is multiplied by a normalization factor since it scales each element separately to the range 0-1, which is the range for floating-point values where there is the most precision. Feature selection, also called subset selection, aim to select a subset of features that can sufficiently represent the characteristic of the original features. The individual signal in the frequency domain contains too many features and thus redundant information. This could cause loss of effectiveness in the predictions of ML algorithms and poor accuracy of the model. To be considered effective it was decided to divide the signal into bins of size of 50 Hz each. For each of these bins the mean and variance were calculated. The average takes into account possible peaks that distinguish important frequency bands. Variance, on the other hand, takes into account the variability of the assumed values within the bins and it is an appropriate index in the identification of fluctuations. As a consequence, there are two vectors, one of the means and the other of the variances of equal size. The two vectors are concatenated in order to obtain a single vector, which characterizes the spectrum of the single signal in a precise way.

5.1. Unsupervised Approach

It is relatively easy to gather training data of situations that are normal; it is just the standard production situation. But on the other side, collection example data of a faulty system state can be rather expensive, or just impossible. If a faulty system state could be simulated, there is no way to guarantee that all the faulty states are simulated and thus recognized in a traditional binary-class problem. To cope with this problem, one-class classification solution is introduced. By just providing the normal training data, an algorithm creates a model of this data. If newly encountered data is too different, according to some measurement, from this model, it is labeled as out-of-class and so anomalous. Nominal data after the pre-processing and feature-selection phases are ready to be consumed by the svm.OneClassSVM module of the Scikit-learn library [5]. The one-class SVM then tries to find a boundary that encloses regions of high data density excluding at most a fraction of data points. It can be seen that boundaries which are linear in the problem variable space are too simple for most problems. Since SVM is a linear classifier by nature, it is needed to resort to kernel methods to build a flexible model with non-linear boundaries. The one-class SVM tries to find a hyperplane separating the data from the origin. The idea is to map the data into the kernel feature space and to separate it from the origin with maximum margin using a linear classifier in the kernel feature space. This corresponds to using a non-linear boundary in the original problem space. The inliers will be on one side of the decision boundary and all the outliers will be on the other side of the decision boundary. In the end, the algorithm detects the soft boundary of the set of samples and then return the one-class model through the fit() function.

5.2. Supervised Approach

When both nominal and anomalous data are available, accompanied by labels that defining them, it is possible to use supervised anomaly detection techniques including Neighbors-based classification. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point. Several nearest neighbors classifiers have been developed to address multi-class classification problems among which, this thesis makes use of neighbors.KNeighborsClassifier module of the Scikit-learn library [5]. It is a classifier implementing the k-nearest neighbors vote and the optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct. In this work is used k equals to 3 and since data belong to the nominal class (+1), synthetic anomalous data has been created from normal data in order to have an anomalous class (-1). Synthetic data is information artificially manufactured rather than generated by real-world events. So a supervised model can be trained with both nominal and anomalous data that went through the pre-processing and feature-selection phases, with the appropriate labels.

6. Implementation

The client can run the Python script passing the appropriate parameters via the terminal command line:

- **service** parameter that can be training or update or inference.
- **path** parameter that can be a path of a Csv file (inference) or a path of a folder containing a series of Csv files (training, update).

Moreover, a configuration file has been written in JSON format that contains the parameters "ip" and "port" relative to the server to make requests. Python script takes the parameters and after the serialization of the data into JSON format, send the GET request containing the JSON in the body, to the server with the appropriate APIs. The server is built with Uvicorn, that is an Asynchronous Server Gateway Interface (ASGI) web server implemented in Python. ASGI not only allows multiple incoming events and outgoing events for the application, but also allows for a background coroutine so the application can do other things. The RESTful APIs, instead are developed in Python with FastAPI web framework that fully supports asynchronous programming and can run with Uvicorn.

7. Amazon EC2

The server code can be deployed into an Amazon Elastic Compute Cloud (Amazon EC2) in order to provide scalable computing capacity in the Amazon Web Services (AWS) Cloud. In the case study addressed in Chapter 8 has been created an Amazon EC2 T3a instance. It provides a baseline level of CPU performance with the ability to burst CPU usage at any time for as long as required. After having created the virtual computing environment, the server is deployed as systemd service and the server code runs with the Uvicorn command. In this way we have obtained a system of Anomaly detection as-a-Service that actually works on AWS machines and able to provide the services.

8. Case Study

The case study addressed in this thesis is that of Eisenmann Italia S.r.l. company. The aim of the Eisenmann Italia S.r.l. company is to build a monitoring system for the identification of anomalies in the perspective of predictive maintenance in Industry 4.0. The sensors used are triaxial accelerometers and they are directly mounted on the machines mentioned above. Sampling periods are carried out alternating with periods of inactivity of the sensors. The type of phenomena on which the work focuses have a constant pattern over time. Therefore it is expected that a machine will have a slow degradation with a certain continuity and not a transient failure. The sensors have a sampling frequency of 5000 Hz. The duration of each sampling is equal to 30 seconds and the sampling rate shall be hourly. The data are stored in a shared database so that they can be accessed and each data sample is stored inside a Csv file. The metadata of the sample is contained in the file name, which is composed of the sensor name followed by the sampling start timestamp (date and time). The Csv file consists of 4 columns: **TIME** that is a counter useful to sort the individual sampling, and the 3 measurements along the **X**, **Y** and **Z** axis. The length of the Csv file is proportional to the sampling frequency and the duration of the sampling. In this case being the sampling frequency equal to 5000 Hz and continuing to sample for 30 seconds, the length of each Csv file will be 150k rows. Eisenmann Italia S.r.l. company provided the Sensor_1 and Sensor_2 data, mounted on different physical systems are available, which are respectively composed of approximately 640 and 300 signals each. These data are obtained from steady-state measurements, they are therefore related to the nominal/correct operation of the systems. Moreover, the company have performed a data quality assessment in fact data have the same formats, the same dimensions and no missing values.

9. Experiments and Results

In order to know the anomalous states of the data and the way in which they present themselves, it is necessary to define their characteristics. The machine on which a sensor is mounted may tilt. This would lead the sensor to record noise or bias that can be equivalent to the addition of: white noise tuned with feasible mean and standard deviation to nominal data, and sinusoids with appropriate magnitudes to nominal data. White noise in the frequency domain occupies the whole spectrum band and will therefore affect the signal in all frequencies. On the other hand, sinusoids in the frequency domain occupy only a limited band of the spectrum, which corresponds to their frequency and will therefore affect the signal with unusual peaks. The production of the synthetic failures is based on the types of anomalies written before. These introductions applied to nominal data in the time domain on each axis (X,Y,Z) lead to undesired effects in the frequency domain.

9.1. Unsupervised model

The unsupervised one-class SVM model relative to Sensor_1 has been trained with 240 nominal samples of Sensor_1. The model has been tested on the remaining 400 samples (of Sensor_1) of which 200 were not modified and 200 were modified at each different configuration with the addition on each axis (X,Y,Z) of:

(a) White noise with mean $\mu = 0$ and standard deviation $\sigma = 0.16$.
(b) White noise with mean $\mu = 0$ and standard deviation $\sigma = 0.3$.
(c) Sinusoids with frequency $1200Hz \leq f \leq 1202Hz$ and magnitude equals to the max of each axis.
(d) Sinusoids with frequency $1200Hz \leq f \leq 1203Hz$ and magnitude equals to the max of each axis.
(e) Sinusoids with frequency $600Hz \leq f \leq 601Hz$ and magnitude equals to the max of each axis.

The training phase and the test phase with all configurations, were repeated 5 times in order to have 5 different random partitions of the data. The accuracy results obtained by the unsupervised model are shown in the Table 1, divided by type of synthetic failure injected. Mean and variance refer to mean and variance of accuracies.

	Partition1	Partition2	Partition3	Partition4	Partition5	Mean	Variance
(a)	0.615	0.6	0.6075	0.975	0.9025	0.74	0.0268825
(b)	1.0	0.995	0.9825	0.9925	0.9925	0.9925	3.2499e-05
(c)	0.6	0.6075	0.575	0.5775	0.685	0.609	0.0016015
(d)	0.9475	0.94	0.975	0.965	0.9575	0.9570	0.000154
(e)	0.4925	0.495	0.505	0.5125	0.5175	0.5045	9.3499e-05

Table 1: Accuracies of unsupervised model

The results of the experiment show that the unsupervised model is able to correctly classify data affected by a relevant white noise. With less evident white noise (lower variance), the accuracy of the model is reduced depending on the partition. As for the data with the addition of sinusoids: the model is able to correctly classify the sinusoids that have peaks in the frequency domain that occupy more than one frequency. On the other hand the peaks that occupy a single frequency are not classified as anomalous by the model. The model therefore does not assess the latter case with a significant weight. In fact the accuracy of the model is higher the more peaks occupy a large frequency band, denoting a unconventional intensity event.

9.2. Supervised model

In the context of supervised approach, the Sensor_1's dataset (640 samples) was split in half randomly. One half (320 samples) left as it was. The other half have been added:

(a) White noise with mean $\mu = 0$ and standard deviation $\sigma = 0.08$.
(b) Sinusoids with frequency $1200Hz \leq f \leq 1202Hz$ and magnitude equals to the max of each axis.
(c) Mix of sinusoids with frequency $600Hz \leq$

$f \leq 601Hz$ and magnitude equals to the max of each axis, and white noise with mean $\mu = 0$ and standard deviation $\sigma = 0.06$.

(d) Sinusoids with random frequencies and magnitude equals to the max of each axis.

The supervised KNN model relative to Sensor_1 has been trained with 240 samples of Sensor_1. The model has been tested on the remaining 400 samples. The split of the dataset, the training phase and the test phase with all configurations, were repeated 5 times in order to have 5 different random partitions of the data. The accuracy results obtained by the supervised model are shown in the Table 2, divided by type of synthetic failure injected.

	Partition1	Partition2	Partition3	Partition4	Partition5	Mean	Variance
(a)	0.95	0.945	0.945	0.9475	0.9475	0.94699	3.50e-06
(b)	0.99	0.9975	0.995	0.9925	0.9575	0.9865	0.000216
(c)	0.8625	0.925	0.9175	0.9025	0.8775	0.89699	0.000561
(d)	0.4625	0.5125	0.5225	0.52	0.4925	0.502	0.0005009

Table 2: Accuracies of supervised model

The results of the experiment show that the supervised model is able to correctly classify data affected by both relevant and less evident white noise. And the accuracy of the model remains high. On the other hand sinusoids with peaks that occupy random frequencies are not classified as anomalous by the model. The model therefore does not assess the latter case with a significant weight. In the scenario in which the case study is carried out, the supervised approach tends to be too limited because it would not be able to correctly classify types of malfunctions that are different from those with which the model has been trained. On the contrary, the unsupervised approach despite the results obtained during the experiments carried out, would seem to be a practical solution that adapts very well to the real need of the company.

10. Conclusions

The thesis project carried out aimed at the design and construction of a client-server architecture for the Anomaly detection as-a-Service via RESTful APIs. This solution is capable of supporting predictive maintenance activities of Eisenmann Italia S.r.l. in Industry 4.0 context. The developed architecture has proven to be able to provide the three services mentioned above. Moreover it classifies and detects anomalies, thus enabling diagnostic functions that reduce losses and total machines failure. In addition, the solution is scalable across multiple sensors and consequently across multiple machines. In the event that the machine on which the sensor is mounted undergoes unnatural movements such as involuntary blows, it will be necessary to collect new data and then re-train the model. This is because otherwise the model would evaluate the current state of the machine as anomalous. The results obtained by unsupervised model, based on the one-class SVM algorithm, are confident for use in industrial processes in the scenario in which Eisenmann Italia S.r.l. is located. On the contrary the use of supervised model, based on the KNN algorithm, does not guarantee reliability on uncharted anomalous states where the machines could be found. In the end, the Anomaly detection as-a-Service is a promising and growing area of research. And the developed client-server architecture can start future improvements and applications in different areas of work.

References

[1] Peter Mell and Timothy Grance. The nist definition of cloud computing, 2011-09-28 2011.

[2] Yong Haur Tay. Xylorix: An ai-as-a-service platform for wood identification, 05 2019.

[3] Uma Subbiah, Muthu Ramachandran, and Zaigham Mahmood. Software engineering approach to bug prediction models using machine learning as a service (mlaas). In *ICSOFT*, 2018.

[4] Smitha Rajagopal, Katiganere Siddaramappa Hareesha, and Poornima Panduranga Kundapur. Performance analysis of binary and multiclass models using azure machine learning. *International Journal of Electrical and Computer Engineering*, 10(1):978–986, January 2020. ISSN 2088-8708. doi: 10.11591/ijece.v10i1.pp978-986.

[5] Gilles Louppe Lars Buitinck. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.