POLITECNICO DI TORINO

Master degree course in Electronic Engineering

Master Degree Thesis

Detection of food contaminants with Microwave Sensing and Machine Learning

Advisors:
prof. Mario Roberto Casu
prof. Francesca Vipiana
dott. Marco Ricci

Candidate:
Luca Urbinati

Academic Year 2018-2019

To my family and friends,
because fundamental
for my entire academic path.

Summary

Today, food contamination due to foreign body is still a trouble for food manufacturers. First of all, because they have to guarantee a safe product to consumers. Secondly because a contaminant can damage the company reputation and lead to expensive recall campaigns. Finally, because more accurate foreign body detection systems allow the producers to gain important food certifications that could increase their incomes.

For these reasons many techniques are currently adopted to solve this problem, such as mechanical filters, metal detectors, X-rays imaging and near infrared imaging. However, there are still some limitations: nonmetallic and low-density objects, like fragments of plastics, glass and wood, are not currently detectable; high penetration depth and low spacial resolution trade-off is still relevant; some methods are subjected to water attenuation.

The good news is that Microwave Imaging Technology has the potential to overcome those problems in addition to other attractive characteristics. Indeed, it is non-destructive, contact-less, non-ionizing, real-time, cost-efficient and easy to operate.

The goal of this thesis is to apply Machine Learning (ML) algorithms to a Microwave Sensing (MWS) system to identify foreign objects in hazelnut-cocoa spread jars on the conveyor belt.

Two kinds of binary classifiers are employed: a Support Vector Machine (SVM) and a Multilayer Perceptron (MLP). The training is performed on two different static and balanced datasets.

The first one consists of 1800 synthetic tomographic images. A 10-folds cross-validation (CV) accuracy of 99.167% is reached for the SVM and a 5-folds CV accuracy of 99.167% is achieved for the MLP, with an error of 0.278% and 1.111% over a test set of 360 samples, respectively. The purpose of this dataset is to validate the idea of applying ML to the foreign body detection problem in the hazelnut-cocoa cream jars with MWI.

Since the results obtained with the first synthetic dataset were encouraging, a second dataset is created. It is composed by 2400 samples of scattering parameters of real measurements, acquired with a MWI system prototype. In this case the results are: 10-folds CV accuracy of 95.052% for the SVM and 5-folds CV accuracy of 95.833% for the MLP, with 6.04% of error over a test set of 480 samples, for both.

To conclude, since both the best SVM and the best MLP give the same error rate on the test set, the latter is chosen for a faster hardware implementation. It is translated into an High Level Synthesis (HLS) C/C++ code and then synthetized for the Xilinx Zynq®-7000 SoC with Vivado HLS. This allows a large and quick design space exploration to satisfy the 100 ms latency requirement of this project, while optimizing area, power and throughput. The classification performances on the same Real Test Set are confirmed and the lowest latency is around 3 ms.

Regarding the future works, the main perspectives are to: enlarge the static real dataset, including also new types of intrusions, such as wood; train different classifiers, especially an ensemble; validate the classifiers with dynamic measurements; discover which is the optimal switching sequence of active antennas pairs during the transit of the jar below the arch to maximize the illumination of the target; validate the behavior of the system on different homogeneous food products, such as honey, yogurt, baby food, and on non-homogeneous ones, like chocolate spreads with hazelnut grains, to have a wide range of possible application scenarios.

Acknowledgements

I wish to thank my advisor for the proposal of this interesting activity, which aims to solve a serious industrial issue. On top of that, I really liked the possibility to experiment on my own and without particular constraints the vast field of Machine Learning. Politecnico di Torino played also an important role because it provided for the necessary resources to carry out the project. Finally, a special acknowledge is for my co-advisor, who constantly listened to my requests.

Contents

Chapter 1

Introduction

1.1 The food contamination problem

Nowadays, food contamination is a serious concern for food manufactures. In fact, there are many reasons for them to adopt new and more sophisticated techniques to detect foreign bodies in their production lines.

First of all, it is better to define what is a foreign body. It is an object or piece of extraneous matter that may accidentally contaminate food [1]. It is extrinsic when come from external sources either by deliberate or accidental means, while it is intrinsic when it is an intrusion that is related to the same food category, but its presence in the final product is a mistake [2]. Examples of the first type could be fragments of plastic, glass, wood, metal, paper, but also stones, insects, human hairs and so on [3]. Instead, regarding the second, they can be bones and gristle in a meat product, a leaf or stalk in a pack of frozen vegetables, or an ingredient in an unusual/unexpected state, like crystallized ingredients considered as glass by consumers [2].

Coming back to the initial statement, the producers' main motivations are the following:

- the potential harm that a foreign body can cause when ingested by the consumer, either due to an injury or to an allergen intolerance [2];
- the subsequent legal battle and claim for compensation;
- the financial and reputational damage, with brand loyalty loss in the public opinion, these days always more often blown up through social media [2] [4], which can vanish years of efforts and investments in advertisements;

- the expensive recall campaign;
- the need of a certain food certification, to elevate the company quality food standard, promising new markets openings and the capability of forcing higher products prices [5].

How is the problem of food contamination currently tackled by manufacturers? Are there some limitations and lacks of controls in their production lines? If so, how can be solved?

1.2 The food contamination solutions

1.2.1 Prevention and control strategies

Prevention plays a key role in food industry: several guidelines are applied during the manufacturing process to attempt to avoid hazards, such as *Good Manufacturing Practice* (GMP) [6] and *Hazard Analysis and Critical Control Point* (HACCP) [7]. However the risk of a foreign body is still present and therefore other strategies have to be introduced.

The control strategies are divided in four categories: *Off-line*, *At-line*, *On-line* and *In-line*. They mainly differ in the proximity to the production line and the analysis time. The first three require the sampling of a product to generalize the results for the whole production batch. Instead, the last controls every item [8]. The first two can also be destructive, i.e. they alter the food under analysis in an irremediable way because chemical reagents are employed, while the last two categories are non-destructive.

1. The **Off-line** approach provides for a laboratory analysis to be done. The place of the lab is outside the industry and requires days to get the results.
2. The **At-line** method involves an examination that is carried out in situ and needs several minutes to complete.
3. The **On-line** procedure is automatic and analyzes the process line every few seconds. The sample can be temporary removed from the main line to be investigated or can enter in a second short and slow line where it is sampled in real-time. After the examination, the sample returns in the main line. The speed of the main line is not altered.

4. The **In-line** mechanism is automatic and inspects every sample (without sampling) in less than a second, always leaving it along the manufacturing chain. The speed of the main line is not altered.

Obviously Off-line and At-line strategies are obsolete for intrusion detection in agri-food products [9]. Therefore Paragraph 1.2.2 deepens On-line and In-line procedures. Instead, for what concerns the main Off-line and At-line methods, only a list of them is reported because they are beyond the scope of this thesis: Nuclear Magnetic Resonance (NMR), Electronic microscopy, Mass spectrometry, Chromatography [3] and others [9].

1.2.2 On-line and In-line hazard detection techniques

Let us report the principal techniques, that can be both applied on-line or in-line, with their advantages and limitations:

- **visual inspection**: operators control the products in terms of shape, color and other qualitative characteristics. This method is human resource expensive, inclined to human errors, not contact-less, but technologically simple [3];
- **mechanical filters**: grids with different designs and hole diameters are placed along the production line to filter out unwanted objects greater than certain dimensions. They are very important not only for the safety of the final good, but also for the health of the machines involved in the manufacturing chain, such as pumps, which could break. They have to be cleaned often, requiring the interruption of the manufacturing chain [5];
- **magnets**: their purpose and their disadvantage are the same as the mechanical filters, but they attract ferromagnetic materials instead of blocking them;
- **metal detectors**: as the name already suggests, they are able to identify only metal foreign objects;
- **X-rays Imaging**: an high-energy and ionizing beam passes though organic materials and is absorbed by high-density tissues, such as metals, bones, stones and only some types of plastics. A detector array captures the non-absorbed particles, resulting in a graphical representation of the object under inspection. Later, a software analyzes the image looking

for defects: the denser the contaminant, the more contrast appears and the easier it is to detect it. Many X-Ray systems are designed to detect certain plastics, but no X-Ray system can detect all plastics [10]. Their limitations are the involuntary exposition of operators and their blindness with respect to low-density foreign bodies (thin glass and most plastics);

- **optical techniques**: based on the reflection of an incident light beam at one or more frequencies, these methods are capable of finding only surface defects and can require multiple frequencies for a specific type of food;

- **Visible/Near Infrared Imaging**: it exploits the different absorption of materials to derive information on the composition of the exposed food. It is fast and safe, but its weaknesses are the very limited penetration depth, the intense absorption in water (so it is not well suited for fresh food) and the application restriction to organic constituents only[3];

- **Hyper/Multi-Spectral Imaging**: exploiting the normal spectroscopy, it scans the object under evaluation spatially (read images over time) and spectrally (acquire images of an area at different wavelengths) to produce multi-dimensional images which are studied to find a contaminant. Although it has great accuracy, it works only in surface, requires an high acquisition time and a large memory, and it is expensive.

- **Terahertz Imaging**: the principle is the same of the infrared imaging, despite that the working frequency is in the range [0.1 - 10] THz. Its limitations are the penetration depth, the attenuation in water, the high instrumentation cost and the slow acquisition time.

- **Ultrasound Imaging**: a low energy pulse of sound vibrating at frequencies between 3 and 30 MHz is transmitted into the investigated object by a transducer probe touching it. The back-scattered signal is listened by the same probe and it is processesed to form images [11]. It is employed for superficial and internal defects detection and it is real-time. The principal disadvantage is the need of a contact between the transducer and the item to inspect.

- **Thermal (Infrared) Imaging**: since all materials emit an infrared radiation, this technology utilizes that radiation to produce an image

of the thermal distribution of the body's surface [12]. Even though it is contact-less, it is expensive and suffers from thermal interference and difficulty to distinguish the low thermal variation produced by contaminants [13].

1.2.3 Microwave Imaging: a promising technology

A complementary and promising technique, aiming to overcome some limitations in current technologies is emerging: *Microwave Imaging* (MWI). (MWI).

It is based on exploiting the intrinsic dielectric difference among materials through low-power electromagnetic waves at microwave frequencies. These microwaves are radiated through a set of antennas, penetrate the object under test non invasively, and are scattered back to the same antennas which record them. Then, the recorded signals are processed with ad-hoc algorithms which provide a spatial map of the dielectric properties of the object under test, that is a *tomographic 3D image*. The last step requires the analysis of the generated image to locate targets or to distinguish different materials. This technology works because two different materials that are in contact create a discontinuity that scatters the incident wave with different intensity according to the contrast of the dielectric properties of those two materials.

MWI has many attractive characteristics. It is non-destructive, contact-less, non-ionizing, real-time, cost-efficient and easy to operate [13]. This technique is different from those already in the market because covers a specific need. Indeed, it is capable of detecting nonmetallic and low-density contaminants, such as fragments of plastics, glass and wood. Moreover it can be tuned to have an appropriate trade-off between penetration depth and spatial resolution and does not suffer of strong absorption in water [3] [9] [14] [15]. The only drawback is the requirement of a dielectric contrast between the background medium and the contaminant: the higher the difference, the greater the scattered field, the clearer the presence of a contaminant.

Hence, adding an in-line MWI-based intrusion detection system to a typical food manufacturing chain could help to further decrease the presence of a foreign body in the end product.

1.3 Thesis focus

The goal of this thesis is to apply *Machine Learning* (ML) algorithms to a *Microwave Sensing* (MWS) system to identify foreign objects in hazelnut-cocoa spread jars during their way in the production line. It is called *Sensing* because in this application the microwave back-scattered signals are exploited to detect the presence of a contamination without image reconstruction.

Two kinds of binary classifiers are adopted: a *Support Vector Machine* (SVM) and a *Multilayer Perceptron* (MLP).

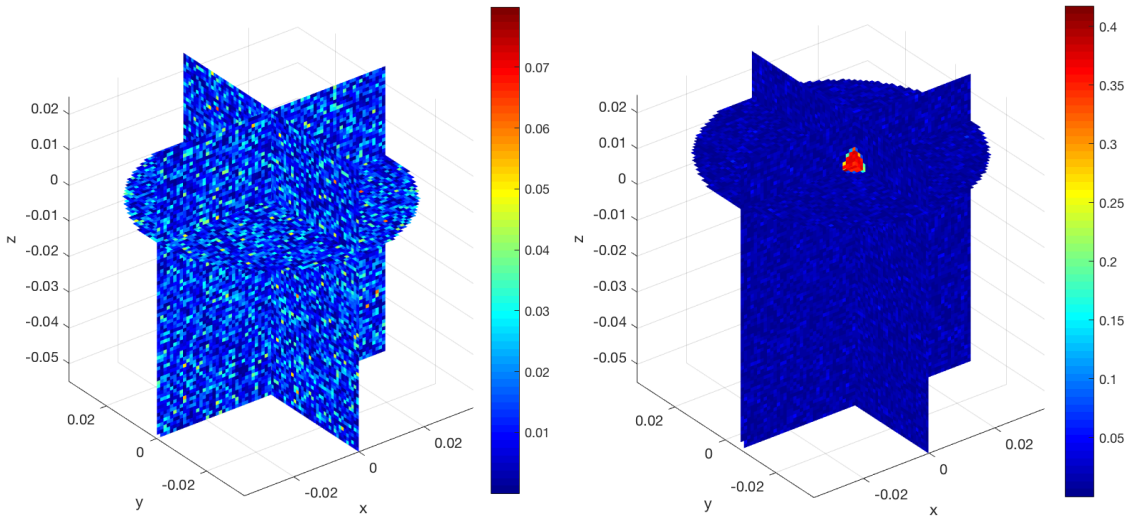
The former is chosen because it is one of the most common ML algorithms for binary classification to start with. In fact, thanks to its two hyper-parameters, it is easy and fast to train and it can help to discover if ML can solve the problem to be addressed [16]. Moreover, it is applied to other detection problems with propitious results. Some examples belong to breast cancer monitoring applications [17] [18] [19].

The latter is selected to have a second classifier to compare with the SVM. Indeed, whenever a new problem is faced with ML, there are no a priori clues about which algorithm will perform better. This is due to the *No Free Lunch Theorem* [20]. As second reason it is considered interesting because of its straightforward hardware implementation in case of satisfactory classification results.

In general, the decision of using these two classifiers was totally unconstrained because in literature there are no specific publications about ML applied to the foreign body detection for industrial food safety.

The training is performed on two different static Datasets, that is the jar is not moving while its acquisition is in progress. In other words, the conveyor belt is off.

The first one consists of 1800 synthetic data. They are tomographic images given by the projection of the imposed dielectric contrast change onto the Truncated Singular Value Decomposition (TSVD) of the linear operator $\mathcal{L}$ of the *reference scenario*, i.e. the golden case in which the jar is uncontaminated [21]. Figure 1.1 shows two tomographic images of an uncontaminated and a contaminated jar, respectively. Before training the two ML classifiers, these data are standardized and reduced in dimensions by Principal Component Analysis (PCA). The purpose of this dataset is to validate the idea of applying ML to the foreign body detection problem in the hazelnut-cocoa cream jars with MWI.



(a) **(b)**

Figure 1.1: Two examples of 3D tomographic images reconstructed with the projection of the imposed dielectric contrast change onto the Truncated Singular Value Decomposition (TSVD) of the linear operator $\mathcal{L}$ of the reference scenario.
The simulated scenario consists in a cylindrical jar, whose base radius is 3.5 cm and its height is 8 cm. In (a) an uncontaminated jar. In (b) a contaminated jar where the presence of the intrusion is clearly visible. The simulated intrusion has a diameter of 1 cm and a dielectric constant of $\varepsilon_{plastic} = 4.1$ @ 10 GHz.

Since the results obtained with the first Synthetic Dataset were encouraging, a second Dataset is created. It is composed by 2400 samples of scattering parameters of real measurements, as proposed by [22] for apple classification. Data are acquired with the *MWI system prototype*, also called *MIT-Food prototype* [23], depicted in Figure 1.2.

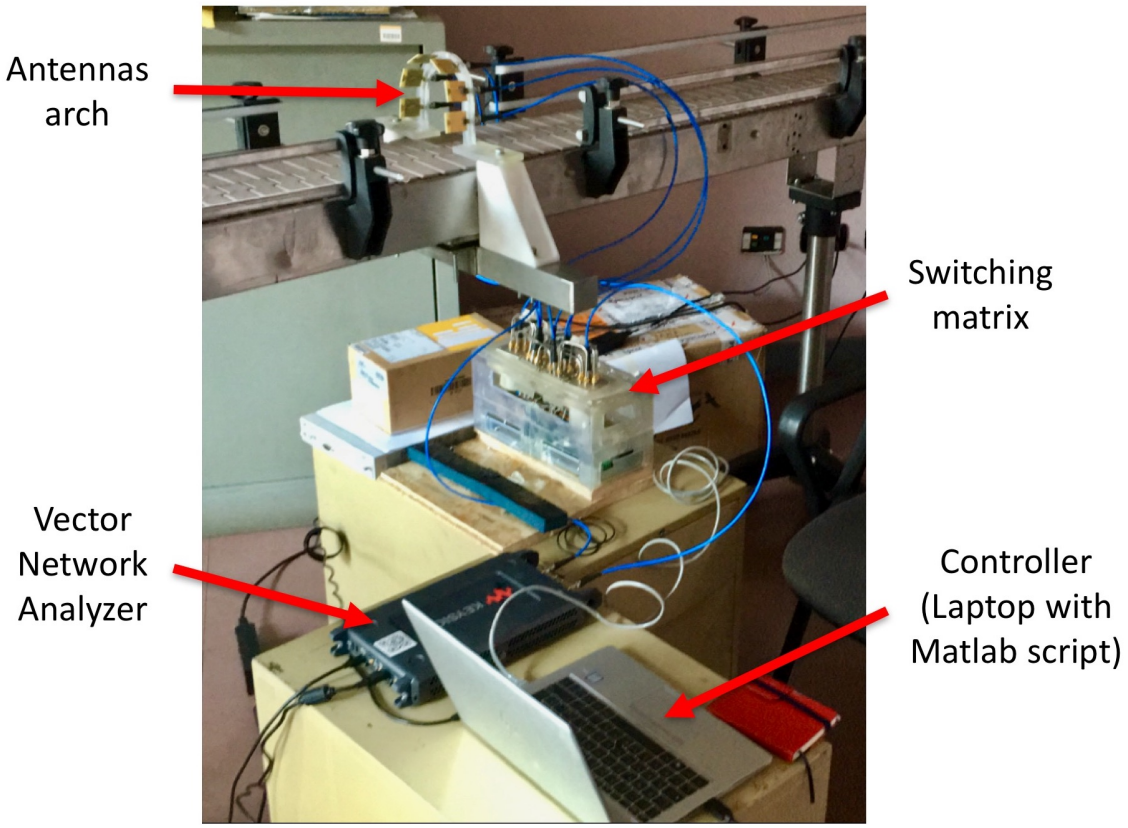
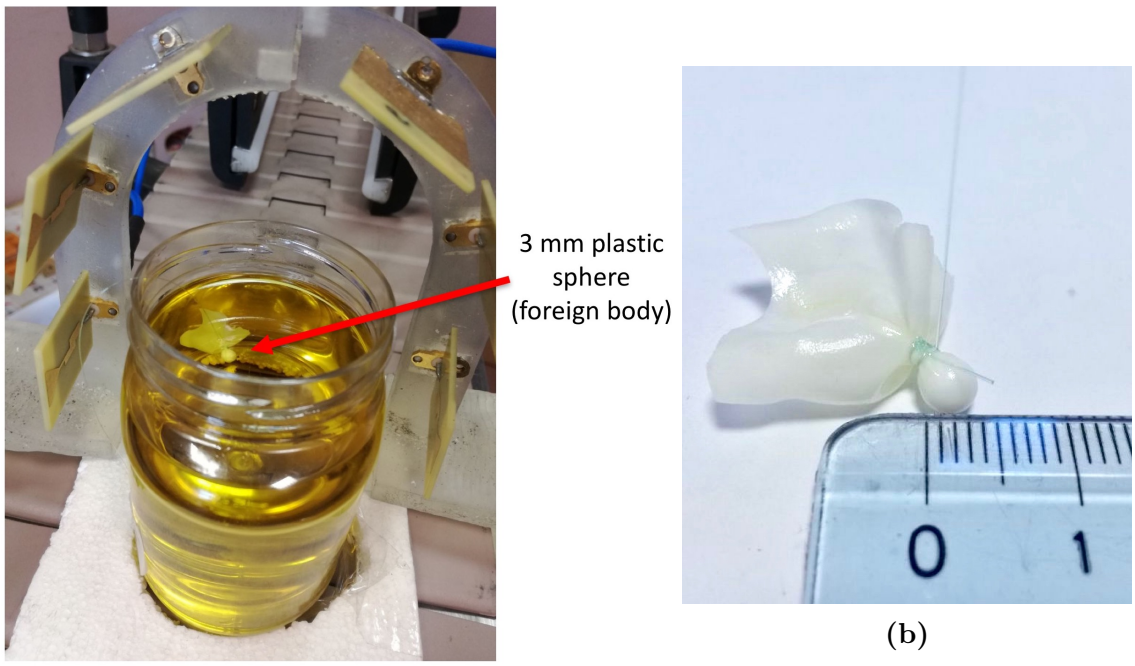


Figure 1.2: Overview of the *MWI system prototype* composed by: an antennas arch, a Vector Network Analyzer, a switching matrix, a controller (laptop with a *Matlab* script).

- **an antennas arch** [21]: a 3D printed support was designed in order to fix six antennas in a gallery-shaped architecture: it aims to offer a multi-view description of the object under test;



(a) **(b)**

Figure 1.3: In (a) the antennas arch with a safflower oil jar beneath it and a foreign body inside. The safflower oil replaces hazelnut-cocoa cream to see the position of the contaminant during the measurements. It doesn't change the dielectric properties of the chocolate because $\varepsilon_{oil} \simeq \varepsilon_{cream} = 2.86$ @ 10 GHz.
In (b) the foreign body: a 3 mm plastic (PET) sphere ($\varepsilon_{plastic} = 3.2$ @ 10 GHz) closed in a latex glove knotted with a fishing wire to handle it easier.

- **a Vector Network Analyzer (VNA)**: it is an instrument which measures the scattering parameters (also called S-parameters) of the network composed by the jar, the antennas arch and the air in between. In this project a two-port VNA is used (one transmitting (TX) and one receiving (RX)), and a switching matrix and a controller are required, since the number of antennas is six;
- **a switching matrix**: it is a device designed to let all the possible couples of antennas in the system interact to get a complete 6×6 Scattering-matrix with the 2-ports VNA;

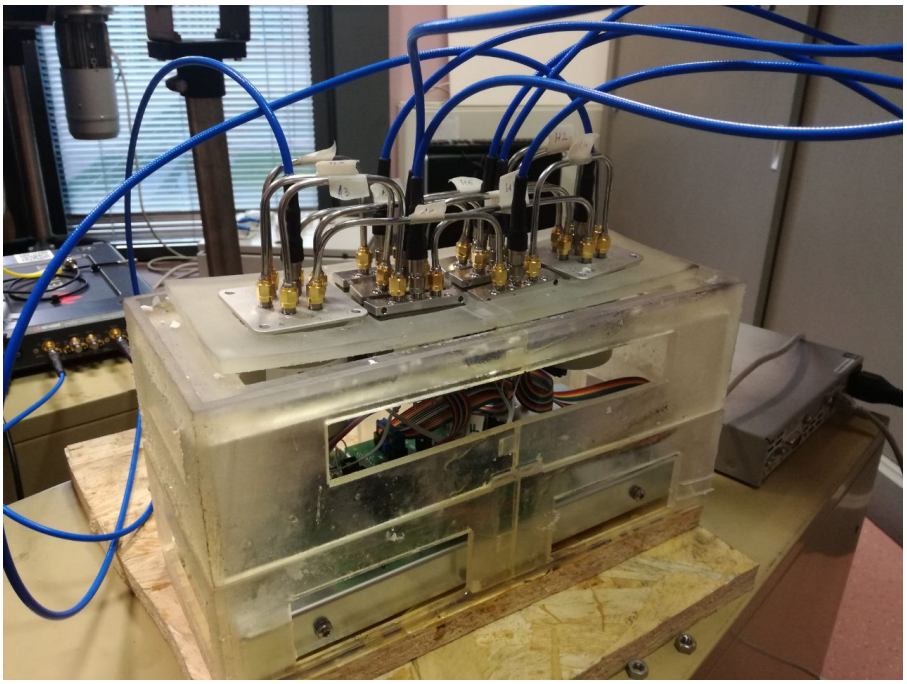


Figure 1.4: Switching matrix with its driver board.

- **a controller**: it is a laptop with a *Matlab* script which pilots the measurements by configuring the switching matrix and by ordering the VNA to acquire the multi-static S-parameters. It is also in charge of saving this S-matrix in a textual format.

Finally, the most performing ML algorithm is synthetized for the *Xilinx Zynq-7000 SoC* to create an hardware accelerator, with the help of Xilinx Vivado tools. This is necessary because:

- it is important to evaluate if the classification performances of the purely software solution are confirmed;
- the *MWI system prototype* of Figure 1.2 has to be added to the production line with a relative compactness;
- the production chain works at a speed of 3 jars per second: assuming that an half-jar space is left between two subsequent jars, the whole system has to classify a jar in 250 ms. A specialized accelerator can be faster than a general purpose processor because it can use specialized arithmetic units, higher level of parallelism and higher degree of pipelining [24]. So latency is the principal constraint of this project;

- the High Level Synthesis (HLS) enables a fast and large design space exploration to satisfy the latency requirement and, at the same time, optimize the other classical hardware constraints, such as area, power, and throughput.

In this way the *MIT-Food system prototype*, combined with ML algorithms, can become the first worldwide In-line prototype based on MWS to detect foreign objects in food/beverage products on a conveyor belt [13].

1.4 Thesis outline

The remainder of this thesis is organized as follows.

Chapter 2 introduces to Microwave Imaging Technology and Machine Learning. In the first part it explains the principal equations which are at the base of MWI to reconstruct the dielectric contrast from the scattering parameters. In the second section, it deals with some basis on ML. In particular, it introduces to SVM and MLP algorithms, those used during the project. Then, it continues with data preprocessing, an important step to extract the best from the input data. In particular, it focuses on Feature Scaling and Feature Extraction, respectively with Standardization and Principal Component Analysis (PCA). Later, it shows the main techniques to train and test ML algorithms, such as Cross-Validation (k-fold approach) and Nested Cross-Validation. Moreover it briefly explains three common techniques for tuning the hyper-parameters of a model: Grid-Search, Random-Search and Bayesian Optimization. In conclusion, the most relevant performance metrics and plots are described: accuracy, precision, recall, confusion matrix, ROC curve, and AUC.

Chapter 3 demonstrates how the Synthetic Dataset (the first of Paragraph 1.3) is generated and preprocessed until the realization of five new datasets. So, SVMs and MLPs are trained on these datasets, first with a loose, then with a fine grid. 5-fold and 10-fold Cross-Validation schemes are used, together with Grid-Search and Bayesian Optimization. Nested Cross-Validation is also employed for the SVMs. Finally, the classification performance of the best models are evaluated on their Synthetic Test Sets. Additionally, the simulated environment is explained, specifically the electromagnetic characteristic of the antennas arch.

Chapter 4 repeats the same steps of the previous chapter applied to the Real Dataset. At the beginning, the specifications of the laboratory instrumentations are given. Next, the creation of the Real Dataset (the second

of Paragraph 1.3) with different types of foreign bodies is illustrated. After its preprocessing, the SVM and MLP classifiers are trained and tested with the same methodologies of Chapter 3 and the most performing models are reported and compared.

Chapter 5 talks about the hardware implementation of the best MLP classifier. First, its Keras model is translated into an High Level Synthesis (HLS) C/C++ code and then it is imported in Vivado HLS. Here it is simulated with the same Real Test Set used in Chapter 4 and the classification performance are confirmed. After a brief design exploration with the constraints of 100 ms as maximum latency and with the target FPGA of the *Avnet ZedBoard Zynq-7000 Development Board*, the promising architectures are implemented in Vivado to estimate their performance, utilization of resources, and power. The one with the lowest latency is the candidate for the industrial application.

Finally, Chapter 6 summarizes the thesis, discusses the relevant results and anticipates the possible future works.

Chapter 2

Microwave Imaging and Machine Learning Theory

The first part of this chapter deals with the algorithm which is at the basis of Microwave Imaging (MWI) Technology. It aims to reconstruct a 3D tomographic image of the dielectric contrast of the object under test from the difference of the Scattering-matrices between the measured scenario and the reference one.

Instead, the second section focuses on the basic topics of Machine Learning and the classifiers employed in this project. It explains the theory behind SVM and MLP algorithms and the meaning of their hyper-parameters. Next, it talks about data preprocessing, in particular Feature Scaling and Feature Extraction, with a special attention to Standardization and to Principal Component Analysis (PCA). It presents the model evaluation schemes that are usually necessary to train and test ML models. Among these the most famous are Cross-Validation and Nested Cross-Validation. Moreover it continues with common techniques for tweaking the hyper-parameters: Grid-Search, Random-Search and Bayesian Optimization. By the end, the metrics used to compare the classification performance of different models are illustrated, from the accuracy, to the ROC curve, passing from the Confusion Matrix.

2.1 Microwave Imaging mathematical background

MWI was introduced in Paragraph 1.2.3, where all its interesting features were discussed in comparison with the other techniques for food inspection. As already said, its goal is to generate a tomographic 3D image of the object under test by exploiting the difference of the dielectric properties among the materials that compose the object itself. In this thesis, the object under test is the hazelnut-cocoa spread jar of Figure 3.2a.

MWI requires a series of antennas around the object under test to measure the multi-static, multi-view Scattering-matrix of the network composed by the object, the antennas, and the air in between. Each p,q-th entry represents the scattering parameter obtained by using antenna p as transmitter (TX) and antenna q as receiver (RX). To evaluate the dielectric contrast change $\Delta\chi$, the difference of the Scattering-matrices ΔS between the *measured scenario* and the *reference scenario* has to be performed. The former is the case in which the jar could potentially contain a foreign body, the latter when the jar is for sure free of intrusions. In this work the antennas are six and they are placed in a gallary-shape structure as shown in Figure 1.3a of Paragraph 1.3 and Figure 4.1 of Paragraph 4.1.1. The result is a 6×6 Scattering-matrix for each acquisition. However, exploiting the reciprocity, the number of measurements could be reduced from $P^2 = 36$ to $P(P+1)/2 = 15$. Therefore, the MWI algorithm has to invert Equation 2.1 in order to find $\Delta\chi$:

$$\Delta S(\boldsymbol{r}_p, \boldsymbol{r}_q) \;=\; \frac{-j\omega\varepsilon_b}{4} \int_D \boldsymbol{E}_b(\boldsymbol{r}_p, \boldsymbol{r}) \cdot \boldsymbol{E}(\boldsymbol{r}, \boldsymbol{r}_q) \Delta\chi(\boldsymbol{r}) d\boldsymbol{r} \tag{2.1}$$

where:

- D is region of interest (ROI), that is the volume of the object under test (the jar);
- $\boldsymbol{r}_p$ and $\boldsymbol{r}_q$ are the positions of the transmitting and receiving antennas, respectively;
- $\boldsymbol{E}_b(\boldsymbol{r}_p, \boldsymbol{r})$ is the "background" electric field radiated in each point $\boldsymbol{r}$ of the ROI by the antenna in position $\boldsymbol{r}_p$ when the volume has no intrusion, so $\Delta\chi(\boldsymbol{r}) = 0$;
- $\boldsymbol{E}(\boldsymbol{r}, \boldsymbol{r}_q)$ is the total field measured by antenna q given by the superposition of the incident field $\boldsymbol{E}_b(\boldsymbol{r}, \boldsymbol{r}_q)$, not dependent of $\Delta\chi$, and the scattered field $\boldsymbol{E}_{scat}$, due to $\Delta\chi$ ($\boldsymbol{E}(\boldsymbol{r}, \boldsymbol{r}_q) \;=\; \boldsymbol{E}_b(\boldsymbol{r}, \boldsymbol{r}_q) + \boldsymbol{E}_{scat}$);

- ε_b is the "background" dielectric constant, i.e. the dielectric constant of the material in the ROI (the hazelnut-cocoa spread);
- ω contains the operating frequency used by the antennas. Its expression is $2\pi f$;
- $\cdot$ is a dot product.

Thanks to the distorted Born approximation [25], which holds until the contrast between the intrusion and the surrounding medium is low and is localized in a small portion of the ROI, $\boldsymbol{E}(\boldsymbol{r}, \boldsymbol{r}_q) \simeq \boldsymbol{E}_b(\boldsymbol{r}, \boldsymbol{r}_q)$. In other words, the contribution of the scattered electric field $\boldsymbol{E}_{scat}$ can be neglected and Equation 2.1 can be linearized in this way:

$$\Delta S(\boldsymbol{r}_p, \boldsymbol{r}_q) \;=\; \mathcal{L}(\Delta\chi) \tag{2.2}$$

where $\mathcal{L}$ is the linear integral operator which relates the aforementioned difference of the Scattering-matrices ΔS to the dielectric contrast change $\Delta\chi$. The kernel of $\mathcal{L}$ is $-j\omega\varepsilon_b/4 * \boldsymbol{E}_b(\boldsymbol{r}_p, \boldsymbol{r}_m) \cdot \boldsymbol{E}_b(\boldsymbol{r}_m, \boldsymbol{r}_q)$, for $\boldsymbol{r}_m \in D$. It is computed off-line for all combination of antennas p and q and for all the positions inside the ROI with Finite Element Method (FEM) simulations.

Next, the Singular Value Decomposition (SVD) of the linear operator $\mathcal{L}$ is performed:

$$\mathcal{L} \;=\; \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^* \tag{2.3}$$

Finally, to reconstruct the the 3D tomographic image, the unknown differential contrast $\Delta\chi$ in each point $\boldsymbol{r}$ of the ROI is calculated by inverting Equation 2.2 and exploiting the Truncated Singular Value Decomposition (TSVD) [25]:

$$\Delta\chi(\boldsymbol{r}) \;= \sum_{n=1}^{N} \frac{1}{\sigma_n} < \Delta S, u_n > v_n \tag{2.4}$$

where:

- ΔS is the variation of the Scattering-matrices between the measured scenario (which can be perturbed or not) and the reference scenario;
- σ_n is the n-th singular value of $\mathcal{L}$, belonging to $\boldsymbol{S} = \{\sigma_n\}$ which is sorted in descending order;

- u_n and v_n are the n-th singular vectors of $\mathcal{L}$, belonging to $\boldsymbol{U} = \{u_n\}$ and $\boldsymbol{V} = \{v_n\}$, respectively, that correspond to the sorted singular values;
- N is the truncation factor that defines the level of information to retain after the TSVD. Its choice is a trade-off between the stability against the noise affecting the measured Scattering-matrix and the accuracy of the reconstructed image [24]. In practice, it represents the number of the couples of antennas to be switched to reconstruct the image;
- $< \cdot >$ is an inner product because $\boldsymbol{U}$ and $\boldsymbol{V}$ are complex numbers vectors.

To see the reconstructed image, $\Delta\chi(\boldsymbol{r})$ is simply plotted in the three-dimensional space. Two examples of the results are reported in Figure 1.1b of Paragraph 1.3: one is a contaminated jar, the other is a free jar. It is important to underline that the more evident is dielectric contrast, the higher is the difference of the Scattering-matrices, the clearer is the presence of a contaminant in the final image.

2.2 Machine Learning basic theory

As the world of Machine Learning (ML) is really wide and researches on this field are still on-going, this paragraph just wants to give a brief introduction to the main topics necessary to understand the subsequent contents of this thesis. In particular, it deals with supervised ML and binary classification.

2.2.1 Support Vector Machine Classifier

Support Vector Machines (SVMs) are born as supervised ML models for binary classification. They can also be adopted for multi-class classification and regression problems. They were first introduced by Cortes and Vapnik in 1995 [26]. Here the attention is on their classification capabilities. The following analysis is taken from [27].

A SVM is based on the concept of decision boundary. Indeed, the training phase has the goal to find the *Support Vectors* (SVs), which are the training samples that define the optimal separating hyperplane, the so called *decision boundary*, that is the hyperplane which separates the samples in the two classes with the maximum margin. The *margin* is defined as the smallest distance between the decision boundary and any of the samples. Figure 2.1

can be helpful to understand these concepts. Moreover, the SVs are used in the prediction phase to assign the class of new unseen data.

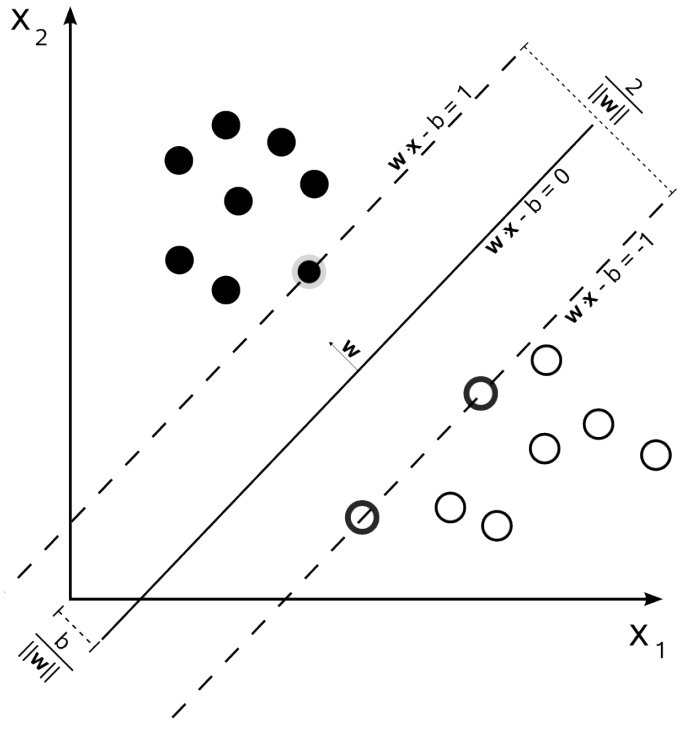


Figure 2.1: An example of optimal separating hyperplane with maximum margin in the Euclidian space. The margin is the distance from the decision boundary $w * x + b = 0$ to the closest sample. The SVs are highlighted dots of both classes which are the nearest to the decision boundary.

Two considerations are important to know. The first is that the smaller is the number of SVs, the lower is the upper bound of the generalization error. The second states that the larger is the margin, the lower is the generalization error of the classifier [20].

Linear SVM

Consider a linear classification problem where the training samples are linearly separable in the feature space. The training dataset is composed by N labelled samples: $(\mathbf{x}_i, y_i),\ i = 1, 2, ..., N,\ x_i \in R^d$ and $y_i \in \{-1, +1\}$, where $\mathbf{x}_i$ is the i-th sample and y_i is the i-th label associated to it. The decision boundary that separates the samples in the positive and in the negative classes is given by:

$$p(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0 \tag{2.5}$$

where $\mathbf{w}^T$ is a vector normal to the optimal hyperplane and b is a constant bias parameter. The training phase wants to find the right choice of $\mathbf{w}^T$ and b such that $p(\mathbf{x}_i) > 0$ for the samples having $y_i = +1$ and $p(\mathbf{x}_i) < 0$ for the samples having $y_i = -1$, that can be summarized in $y_i p(\mathbf{x}_n) > 0$ for all the training data $i = 1, 2, ..., N$. So the decision function of the SVM classifier is defined as:

$$d(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b) \tag{2.6}$$

As the perpendicular distance of a generic point $\mathbf{x}$ from an hyperplane defined by $p(\mathbf{x}) = 0$ is given by $|p(\mathbf{x})|/||\mathbf{w}||$, the distance of the n-th sample $\mathbf{x}_n$ to the decision surface is:

$$\frac{y_n p(\mathbf{x}_n)}{||\mathbf{w}||} = \frac{y_n(\mathbf{w}^T\mathbf{x} + b)}{||\mathbf{w}||} \tag{2.7}$$

in which $p(\mathbf{x}_n)$ has been substituted with its complete expression taken from Equation 2.5. If $\mathbf{x}_n$ becomes the closest point to the decision boundary, the perpendicular distance of Equation 2.7 becomes the margin. Hence, to find the maximum margin solution, Equation 2.7 has to be maximized. This optimization problem can be simplified throughout a rescaling which ends up in setting its numerator $y_n(\mathbf{w}^T\mathbf{x} + b) = 1$. Thus, the optimization problem is simplified in:

$$\arg\min_{\mathbf{w},b} \ \frac{1}{2}||\mathbf{w}||^2 \tag{2.8}$$

where the maximization of $||\mathbf{w}||^{-1}$ has been converted to the minimization of $||\mathbf{w}||^2$.

Thanks to the aforementioned simplification, it is also possible to write the canonical representation of the decision hyperplane that all training samples have to satisfy:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \ n = 1, 2, ..., N \tag{2.9}$$

So far, it was assumed that the final result would have been an exact separation hyperplane. However, since an exact separation of the training data can lead to poor generalization, it is accepted that some training points are misclassified in a controlled way. So the training process has to both maximize the margin and reduce the number errors at the same time. Therefore, a penalty is associated to each n-th data point, called slack variable $\xi_n \geq 0$,

which depends on the position and the distance of each sample with respect to the decision boundary:

- $\xi_n = 0$: the sample is correctly classified, that means it is on the correct side of the decision boundary and it is outside the margin;
- $0 < \xi_n \leq 1$: the sample is inside the margin, even thought it is on the correct side of the decision boundary;
- $\xi_n > 1$: the sample is misclassified because it is on the wrong side of the decision boundary.

The first consequence is that the classification constrains in Equation 2.9 is replaced with:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n, \;\; n = 1, 2, ..., N \tag{2.10}$$

which implies that the exact margin of before is transformed in a *soft margin*. The second is that the minimization problem in Equation 2.8 becomes relaxed:

$$\arg\min_{\mathbf{w},b} \;\; C\sum_{n=1}^{N}\xi_n + \frac{1}{2}||\mathbf{w}||^2 \tag{2.11}$$

where $C > 0$ is a regularization hyper-parameter because it controls the trade-off between minimizing the training errors and controlling the model complexity. Since the term $C\sum_{n=1}^{N}\xi_n$ is fixed, a large C makes the decision boundary more complex because the minimization in Equation 2.11 requires more efforts. In other words, a low C makes the decision surface smooth, while a high C aims at classifying all training samples correctly by complicating the decision surface.

The quadratic programming problem reported in Equation 2.11 is solved by maximizing the dual representation of the Lagrangian function:

$$max \;\; L(\mathbf{a}) = \sum_{i=1}^{N} a_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j a_i a_j \mathbf{x}_i^T \mathbf{x}_j \tag{2.12}$$

subjected to:

$$0 \leq a_n \leq C \tag{2.13}$$

$$\sum_{i=1}^{N} a_i y_i = 0 \tag{2.14}$$

$i = 1, 2, ..., N$. An interesting property of SVMs is that the determination of the model parameters is a convex optimization problem, and so any local solution is also a global optimum.
The solution provides $\mathbf{a}$ which is used to calculate $\mathbf{w}$ and b:

$$\mathbf{w} = \sum_{i=1}^{SV} a_i \mathbf{x}_i y_i \tag{2.15}$$

$$b = y_i - \mathbf{w}^T \mathbf{x}_i \tag{2.16}$$

After the training, and so when the minimization problem is solved, the classification of a new data $\mathbf{x}_{new}$ can be performed by evaluating Equation 2.5 because $\mathbf{w}$ and b are known at this time:

$$d(\mathbf{x}_{new}) = sign\left(\sum_{i=1}^{SV} a_i y_i \mathbf{x}_i^T \mathbf{x}_{new} + b \right) \tag{2.17}$$

where SV is the number of SVs because only the SVs appear in the summation (having $a_n \neq 0$). They are called in this way because they satisfy the equality of Equation 2.9: $y_i p(\mathbf{x}_i) = 1$. The practical consequence is that, once the model is trained, only the SVs are retained, while the other training samples can be discarded.

Non-linear SVM
In many real-world classification problems, it is not possible to linearly separate the training data in the original feature space. In these cases, exploiting the Mercer's condition [26], the input space is mapped to a sufficiently higher dimensional space where a linear separation is feasible. This transformation is carried out with an appropriate nonlinear mapping function $\varphi(\cdot)$ applied to the data $\mathbf{x}$. So the decision function of a non-linear SVM becomes:

$$\begin{aligned} d(\mathbf{x}) &= sign\left(\sum_{i=1}^{SV} a_i y_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}) + b \right) = \\ &= sign\left(\sum_{i=1}^{SV} a_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right) \end{aligned} \tag{2.18}$$

where:

$$b = y_i - \sum_{i=1}^{SV} a_i y_i K(\mathbf{x}_i, \mathbf{x}) \tag{2.19}$$

in which $K(\mathbf{x}_i, \mathbf{x}) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x})$ is called *kernel function*. The most common kernel designs [28] are:

- Linear: $< \mathbf{x}_i, \mathbf{x} >$
- Polynomial: $(\gamma < \mathbf{x}_i, \mathbf{x} > +r)^d$
- Sigmoid: $tanh(\gamma < \mathbf{x}_i, \mathbf{x} > +r)$
- Gaussian Radial Basis Function (RBF): $e^{(-\gamma ||\mathbf{x}_i - \mathbf{x}||^2)}$

where $< \cdot >$ is an inner product and $|| \cdot ||^2$ is the L2-norm.

For this thesis project, the RBF kernel is employed. So this SVM has two hyper-parameters, the regularization parameter C and the kernel-specific γ. The meaning of the first is already discussed above, while the second defines the inverse of the radius of influence of the samples that are selected as SVs during the training [28]. Let us examine the two extreme cases. Basically, if a sample is selected to be a SV and γ is too high, the region of influence defined by that SV is concentrated around it: this causes overfitting, which can't be avoided even by decreasing C to simplify the model. On the other hand, a too small value of γ, makes the region of influence of that SV as wide as to arrive to include the whole training set: the resulting model is like a linear model that separates the two classes with a linear hyperplane. Obviously, reasonable values stays in between these two extremes[1].

It is a common practice to use Grid-Search for the hyper-parameters selection of SVMs because they have only two hyper-parameters to tune [29]. Moreover, it could happen that among the solutions found with Grid-Search some of them have the same value of γ but different values of C. It is preferable to choose the solution with the lower C to keep a simple decision boundary, prevent overfitting, and have a good generation on new unseen data. Finally, in terms of time, a small C reduces training time and speeds up predictions [28].

2.2.2 Multilayer Perceptron Classifier

Multilayer Perceptrons (MLP) are supervised learning algorithms that realize a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where $m = n_{features}$

[1]To visually understand the behavior of the decision boundary of a SVM with Linear and RBF kernels, the reader is invited to visit this website: `https://cs.stanford.edu/people/karpathy/svmjs/demo/`. Accessed 02/12/2019.

is the number of dimensions of the generic input $\mathbf{x}$ and $o = n_{outputs}$ is the number of dimensions of the output. MLPs can become non-linear function approximators for either classification or regression ML problems. They are a variant of the original Perceptron model proposed by Rosenblatt in the 1950 [30]. The following analysis is taken from [28].

The MLP architecture has a fully connected-design. It is divided in layers which are composed by series of *neurons*, also called *nodes* or *units*. Inside a j-th neuron of a generic hidden layer a non-linear function $g(\cdot) : R \to R$ is implemented, named *activation function*, which transform the incoming vector $\mathbf{x}$ from the previous layer with a weighted linear summation:

$$z'_j = g(\sum_{i=1}^{n} w_i x_i + b_j) \tag{2.20}$$

where n is the number of nodes of the previous layer, w_i is the i-th *weight* of the current layer, b_j is the *bias* of the j-th neuron, and z'_j is the output of j-th neuron. In turn, z'_j becomes the x_j of the next layer, etc., in case of a multi-layer structure.

In Figure 2.2 an example of MLP with one hidden layer is shown. In a MLP the minimum number of layers is three, where the leftmost is known as *input layer*, the rightmost is called *output layer*, and all the other layers in between these two are the *hidden layers*. For this basic case, the non-linear function learned by the hidden layer is (in vector notation):

$$z(\mathbf{x}) = \mathbf{w}_2\, g(\mathbf{w}_1^T \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{2.21}$$

where $g(\cdot)$ is the activation function of the neurons in the hidden layer; $z(\mathbf{x})$ is the output of the hidden layer; $\mathbf{w}_1$ and $\mathbf{w}_2$ are the weight vectors of the input layer and hidden layer, respectively; $\mathbf{b}_1$ and $\mathbf{b}_2$ represent the bias added to all the nodes of the hidden layer and output layer, respectively.

Regarding the depth of the input layer, it is equal to $n_{features}$. Instead, in the case of a MLP used for the classification purposes, the number of output nodes $n_{outputs}$ usually corresponds to the number of classes $n_{classes}$ that the model has to classify, except for the binary case in which one output neuron is enough. Typically in a multi-class situation the activation of the output layer is a *softmax* function, whose expression is:

$$h(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{l=1}^{n_{classes}} e^{z_l}} \tag{2.22}$$

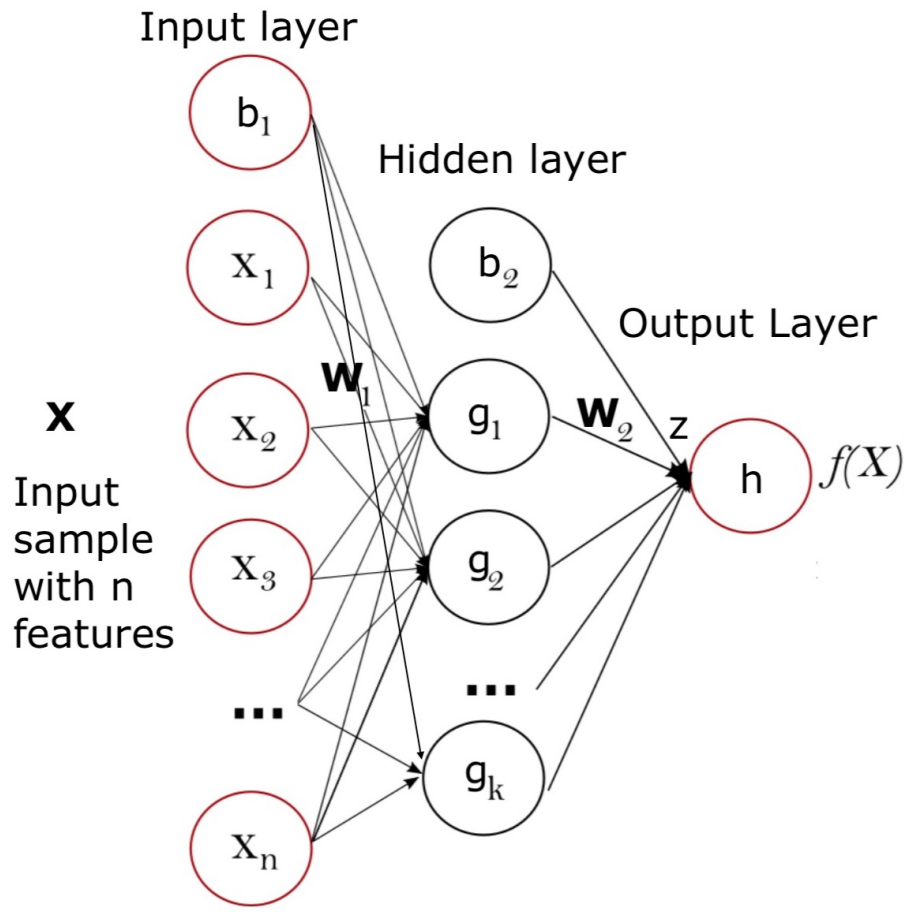


Figure 2.2: Example of MLP with one hidden layer.

where z_i is the i-th element of the input vector $\mathbf{z}$ and $h(\mathbf{z})_i$ is the i-th element of the output vector of the softmax function. $h(\mathbf{z})_i$ represents the probability that the input sample $\mathbf{x}$ belongs to the i-th class.

Instead, for binary classification, the output node performs a *logistic function*, i.e. a *sigmoid*, whose output is a scalar between 0 and 1:

$$h(z) = \frac{1}{(1 + e^{-z})} \tag{2.23}$$

where z is again the weighted linear summation of Equation 2.21 for the case of Figure 2.2. The default discrimination threshold for the result of the sigmoid function is set to 0.5 by default: when it is greater or equal 0.5, the input sample is classified belonging to the positive class; otherwise, it is predicted to be negative.

The vector of the weights $\mathbf{w}$ and the vectors of the biases $\mathbf{b}$ of Equation 2.21 are the parameters that the model has to learn during training. A common choice to find them is by minimizing the training error, and so the *Loss Function*. Several loss functions are available depending on the type of problem. For classification it is the *Cross-Entropy* that becomes *Binary Cross-Entropy* for the binary case:

$$Loss(\hat{y}, y, \mathbf{w}) = -y \ln \hat{y} - (1 - y) \ln (1 - \hat{y}) + \alpha R(\mathbf{w}) \tag{2.24}$$

where y is the label associate to the input sample that is processed by the MLP, $\hat{y}$ is the predicted label, R is a regularization term that penalizes model complexity, $\alpha > 0$ is an hyper-parameter that controls the magnitude of the penalty (also called *Weight Regularization Parameter*).

The common choices for the regularization term R are:

- L2 norm: $R(w) = \frac{1}{2} \sum_{i=1}^{n} w_i^2$;
- L1 norm: $R(w) = \sum_{i=1}^{n} |w_i|$;
- Elastic Net: $R(w) = \frac{\rho}{2} \sum_{i=1}^{n} w_i^2 + (1 - \rho) \sum_{i=1}^{n} |w_i|$, a convex combination of L2 and L1, where ρ controls that combination.

To minimize the Loss Function there are different algorithms available, referred to as *optimizers* in Keras. Two of the most famous are *Stochastic Gradient Descent* (SGD) [28] and *Adaptive Moment Estimation* (Adam) [31]. The SGD iterates over the training samples and for each of them updates the model parameters $\mathbf{w}$ according to the following update rule:

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \eta \nabla Loss(\hat{y}, y, \mathbf{w})^i \tag{2.25}$$

where $\eta > 0$ is the *learning rate*, which controls the step-size of the update of the parameters; i is the iteration step; $\nabla Loss(\cdot)$ is the gradient of the loss with respect to the weights $\mathbf{w}$. The parameters $\mathbf{b}$ are updated similarly, but without regularization in the loss function. While η is an hyper-parameter when using SGD, it is not for Adam. In fact, an interesting characteristic of Adam is the ability to adapt the learning rate η automatically, also reducing it with the passing of training time [31]. This contributes to eliminate the learning rate from the list of hyper-parameters to manually tweak.

From Equation 2.25 is evident that to compute the first iteration, the weights have to be initialized. Since the MLP has a non-convex loss function where more than one local minima exist, different random weight initializations can lead to different validation accuracy [28]. Therefore, there are

various *weight initializers* that can be chosen. They can initialize the weights with zeros, ones, a single constant for all weights, random values from a normal distribution, etc. In literature more complex initializers proved to be more effective: *He Normal* and *Lecun Normal* seem to work very well. The reason is that they prevent a learning slowdown because the standard deviation of their Gaussian distribution used for weight initialization is divided by the square root of the fan-in, i.e. the number of input units in the weight tensor [16] [32]. Even so, understanding these dependencies is still a subject of ongoing research, but for sure MLP are very sensitive to the initialization strategy.

In summary, after the weight initialization, the MLP for binary case minimizes the Binary Cross-Entropy loss function of Equation 2.24 by repeatedly updating the weights with Equation 2.25, assuming SGD is used. After having computed the loss, a backward pass propagates it from the output layer to the previous layers, assigning an update to each weight proportional to the error computed in the output. The final goal is to decrease the loss at the next iteration. The computation of the gradients and the propagation of the errors for the weight updates is carried out by an algorithm called Backpropagation [16].

In this last part the remaining hyper-parameters, usually considered when training a MLP, are briefly described.

- **Number of hidden layers & hidden units**. These are the two main concerns for a designer because these options define the architecture of the MLP. In literature there are so many different rules of thumbs based more on experience than on science that complicate the choice of these two hyper-parameters. Each ML book provides its own version. For example, according to [20], three layers suffice to implement any arbitrary function and only for special problem conditions or requirements more than three hidden layers should be considered. Moreover, networks with multiple hidden layers are more susceptible to face unwanted local minima. So, when facing a new ML problem, it is common to start with a single hidden layer. Since the question is still open because every ML problem is different from the others, a secure alternative is to make experiments. Spanning many solutions with smart hyper-parameters tuning techniques, such as Bayesian Optimization (Paragraph 2.2.5), can be a solution.

- **Activation functions**. Neurons can compute many activation functions as long as they are non-linear. When approaching a ML problem with a MLP, the most used are:

$$elu : s(x) = \begin{cases} x & if\ x > 0 \\ \alpha e^x - 1 & if\ x < 0 \end{cases} \tag{2.26}$$

$$relu : s(x) = max(x, 0) \tag{2.27}$$

$$selu : s(x) = k \cdot elu(x, \alpha) \tag{2.28}$$

$$tanh : s(x) = \frac{e^x - e{-}x}{e^x + e{-}x} \tag{2.29}$$

$$softsign : s(x) = \frac{x}{|x| + 1} \tag{2.30}$$

$$\tag{2.31}$$

  where k in selu is a scalar constant.

- **Dropout rate**. It belongs to a set of techniques that aim to mitigate the effects of overfitting. Among these there is also the penalty parameter α of Equation 2.24. The idea is simple: during training a group of randomly selected neurons on a specific layer are deactivated. This improves the generalization because stimulates the layer to learn the same input pattern with the remaining neurons. During the prediction phase the dropout is deactivated.

- **Batch-size**. It is the number of samples in a batch, that is the set of samples used in one iteration of model training, which corresponds to one gradient update [33]. In other words, the batch-size sets the number of samples the network "sees" before to update its weights. For example, the batch size of SGD is 1.

2.2.3 Dataset preprocessing

Data are the key elements for ML: it is more likely that a simple model trained with "quality" data[2] outperforms a complex model trained on meaningless

[2]A possible definition of "quality" can be that of a dataset in which: there are no label errors, the features have low noise, the data has been scaled to adapt to the model, the outliers[3] are handled correctly, the training set is representative of the data available to the model at prediction time, etc.

data [33]. In addition, good training data can ease the task of a model and allows it to exploit all its potentiality by learning as much as it can from the input samples. For these reasons, before to train a ML algorithm, it is common to carry out these two activities: *Feature Scaling* and *Feature Reduction*.

Feature Scaling

Its consists in transforming the features of the input samples to be on a similar scale, such that the ML model does not prefer one feature over the others [20]. This could happen when there are some features that have greater numeric range than others. If a model is trained with this kind of data, parts of its calculations could have numerical problems. One example is the inner product between the feature vectors in linear, polynomial or sigmoid kernels of SVMs [29]. Another example concerns MLPs, in which non-uniform features prevent the so called *uniform learning*. When this occurs, the weights associated to larger features reach their final equilibrium values before the others, leading to an higher error rate [20]. Scaling also helps the Gradient Descent algorithm to converge more quickly. In this sense, both SVMs and MLPs benefit from Feature Scaling: the overall performance rises and the training stability improves [33]. To conclude, to apply effectively Feature Scaling, the same scaling method needs to be adopted for both training and test data [29].

Now four common techniques taken from [33] are explained. $\mathbf{x}$ will be referred as the input vector, while $\mathbf{x}'$ will be its scaled version.

- **Scaling to a range**

$$\mathbf{x}' = \frac{\mathbf{x} - x_{min}}{x_{max} - x_{min}} \tag{2.32}$$

  This method can be used when the input data $\mathbf{x}$ is bounded between a maximum (x_{max}) and a minimum value (x_{min}), there are few or no *outliers*[3], and the data will be approximately uniformly distributed across the new range. Ordinary choices for the range of values of the scaled vector $\mathbf{x}'$ are [0, 1] or [-1, 1].

[3]Outliers: values distant from most other values. Regarding input data, are those values that are more than roughly 3 standard deviations from the mean [33].

- **Clipping**
  Simple method that can also be applied before or after other scaling techniques. It is employed when data contain many outliers. Clipping collapses all the feature values above/below a certain maximum/minimum value to a fixed value.

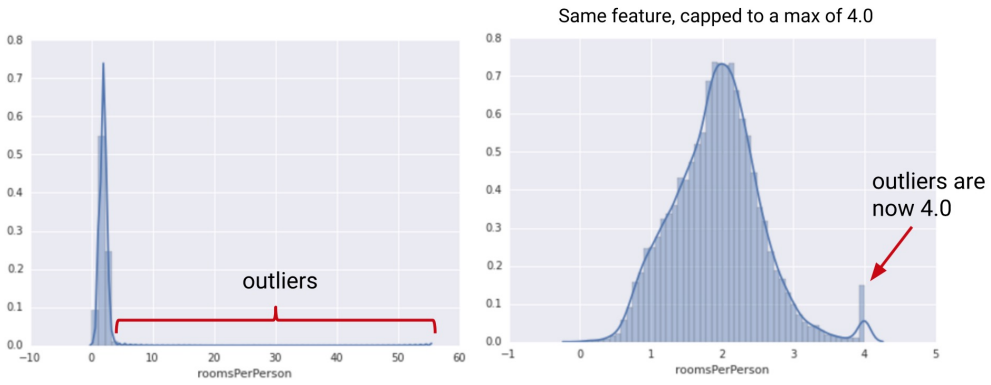


Figure 2.3: Example of the application of the clipping on an initial data distribution with many outliers. Image taken from [33].

- **Log scaling**

$$\mathbf{x}' = log(\mathbf{x}) \tag{2.33}$$

  It uses the logarithmic nature to compress a broad range of feature values to a narrow range. It is useful when data has a power law distribution, that is when there are many samples concentrated in a short range and few samples span over a wider range.

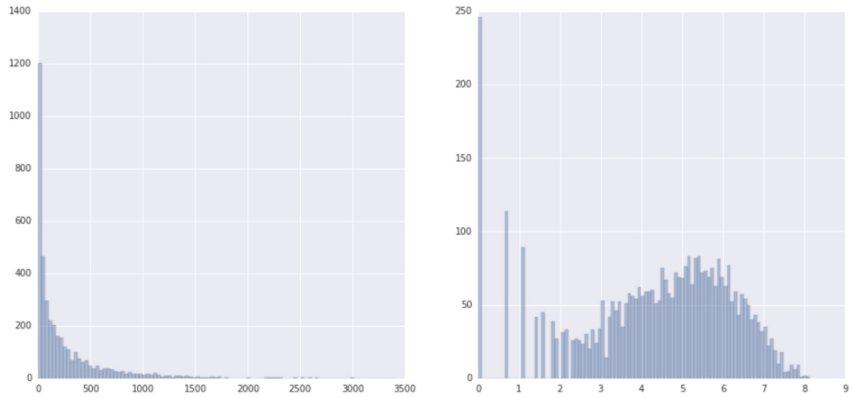


Figure 2.4: Example of the application of the log scaling on an initial dataset with power law distribution. Image taken from [33].

- **Standardization**
  This procedure has different names, such as Z-Score or "centering and scaling". It transforms individual features in standard normally distributed data, like the famous Gaussian distribution with 0 mean and unit variance. Centering and scaling is applied independently on each feature value by computing the relevant statistics on the samples in the training set [28]. It is useful when there are a few outliers and the situation is not so extreme that clipping is needed. The formula is as follows:

$$x'_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j} \tag{2.34}$$
$$\forall i \in 1, 2, ..., n_{samples}, \ \forall j \in 1, 2, ..., n_{features}$$

  where $x_{i,j}$ is the j-th feature of the i-th input vector $\mathbf{x}_i$ to transform, μ_j is the mean of all the j-th features belonging to the training set, σ_j is the standard deviation of all the j-th features of the training set and $x'_{i,j}$ is the j-th feature of the i-th transformed vector $\mathbf{x}'_i$.

Feature Extraction

The goal of this technique, also known as *Feature projection*, is to transform data from an high-dimensional space to a low-dimensional one. Thus, long data samples can be compressed by reducing the number of features without loosing to much information [34]. The reasons behind the application of Feature Extraction are at least three.

The first resides in the fact that having thousands of features complicates the training of the ML models [29] which can result in a very long training time.

The second is related to their prediction performance, which can benefit of a space with reduced dimensionality. In fact, the *Curse of Dimensionality* [35] states that the available data become sparse when the feature space is high-dimensional. This complicates the job of some ML algorithms that want to separate the samples with similar properties. The solutions to this problem are either to increase the number of training samples or to reduce the number of features.

The third is more practical and is related to resource limited applications. Indeed, the elimination of useless features can dramatically reduce the dataset size in terms of disk space occupation. The resulting free space can potentially be used to increase the dataset.

There are plenty of transformations, linear and non-linear [36]. However, the focus of this part is on *Principal Component Analysis* (PCA) because it is needed in the next paragraphs.

- **Principal Component Analysis**
  Assume to have a dataset **D** organized as matrix of ($n_{samples}$, $n_{features}$) with each sample **x** being a vector with length $n_{features}$ which is the number of dimensions. Four steps are required to perform PCA [34].

  1. **Subtract the mean from each feature of each sample**. The subtracted mean μ_j for all the j-th features of all samples is the mean of all the j-th features across all dataset **D**.

  $$x'_{i,j} = x_{i,j} - \mu_j \qquad (2.35)$$
  $$\forall i \in 1, 2, ..., n_{samples},\ \forall j \in 1, 2, ..., n_{features}$$

  where $x_{i,j}$ is the j-th feature of the i-th vector **x** to transform and $x'_{i,j}$ is the j-th feature of the i-th transformed vector **x′**. Let us call the resulting dataset **D'**.

  2. **Compute the covariance matrix**. Since the covariance is calculated between 2 dimensions, if **x** has more than 2 dimensions (i.e. $n_{features} \geq 2$), there is more than one covariance measurement. So all possible covariance values are organized in a matrix, in which each (p,q)-th entry is the covariance between dimension p and dimension q. That matrix is called *covariance matrix*. The definition

of each entry for the general case of data with $n_{features}$ dimensions is:

$$c_{p,q} = cov(Dim_p, Dim_q) \quad \forall p, q \in 1, 2, ..., n_{features} \tag{2.36}$$

where Dim_p and Dim_q are the p-th and q-th column vectors of dataset $\mathbf{D}$ and so have a length equal to $n_{samples}$. Instead, the $cov(\cdot)$ function is defined as:

$$cov(Dim_p, Dim_q) = \frac{\sum_{i=1}^{n_{samples}} (Dim_p - \overline{Dim}_p)(Dim_q - \overline{Dim}_q)}{(n_{samples} - 1)} \tag{2.37}$$

where $\overline{Dim}_p$ and $\overline{Dim}_q$ are the mean of Dim_p and Dim_q.
Here an example of covariance matrix for a 3 dimensional dataset with the usual x, y, z dimensions:

$$\mathbf{C} = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix} \tag{2.38}$$

In general, the covariance matrix is square ($n_{dimensions} \times n_{dimensions}$) and symmetric ($cov(a,b) = cov(b,a)$).

3. **Generate the feature vector**. The eigenvectors and eigenvalues of the covariance matrix are calculated and the eigenvectors are sorted in descending order by eigenvalue. The sorted eigenvectors are organized in a matrix of vectors called *feature vector* (F_V).

$$\mathbf{F_V} = (eig_1, eig_2, ..., eig_{n_{features}}) \tag{2.39}$$

At this point, in order to reduce the dataset dimension, only the eigenvectors associated with the highest eigenvalues are kept. The remaining eigenvectors are referred to as the *principal components* because they retain the most of the dataset information. This is clearly a lossy operation, but the smaller the eigenvalues of the discarded eigenvectors, the less information is lost. Hence, the resulting feature vector becomes:

$$\mathbf{F'_V} = (eig_1, eig_2, ..., eig_{n_{principal\,components}}) \tag{2.40}$$

where $n_{principal\,components} \leq n_{features}$.

4. **Derive the reduced dataset**. The centered dataset **D'** ($n_{samples}$, $n_{features}$) is project onto the new low dimensional dataset **D"** ($n_{samples}$, $n_{principal\,components}$) thanks to this truncated transformation:

$$\mathbf{D''} = \mathbf{D'}\ \mathbf{F'_V} \tag{2.41}$$

The principal components are now the dimensions of the new dataset and they are uncorrelated respect to the initial dimensions.

Sometimes, PCA is also used to visualize an high dimensional dataset in 2 or 3 dimensions by retaining only the first two or three principal components. Three examples can be seen in Figure 3.8, and in Figure 4.6a and 4.6b.

2.2.4 Model evaluation schemes

Training and testing ML models can be done in different ways. Every approach provides a performance score as output, but not always it is possible to generalize the result. In fact, it could depend on the particular bench of data that is used to get that result, both during the training and the test phases. If this happens, those results are not representative of the performance that the model will have on new unseen data, i.e. the model does not generalize. Unfortunately, having a low generalization error requires a complex training and test procedure, which also increases the total training time. This paragraph deals with the most common methods to evaluate ML models in an incremental order of complexity, training time and generalization capabilities. Models trained and tested with different schemes can't be compared between each other.

Model evaluation with held-out test set

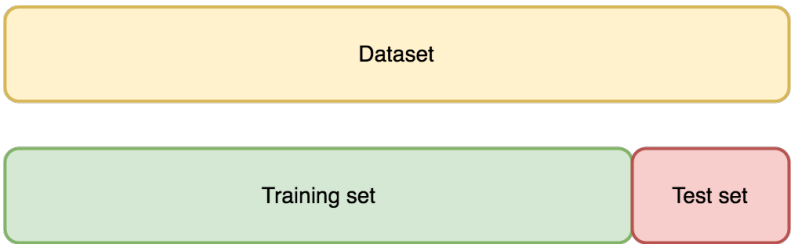


Figure 2.5: Model evaluation with held-out test set schema.

This approach splits the dataset in a training set and a test set. The first part is used to train the model with a certain combination of hyper-parameters.

Instead, the purpose of the second part is to estimate the score (such as the accuracy) and the generalization error of the trained model [20]. The problem of this solution is a serious risk of overfitting on the held-out test set because the hyper-parameters can be tuned until the estimator performs optimally. Moreover, this solution has an high variance together with an high generalization error on the test set because the resulting scores depend on the specific data that compose both the training and test splits. This method should be never adopted.

Model evaluation with held-out validation and test sets

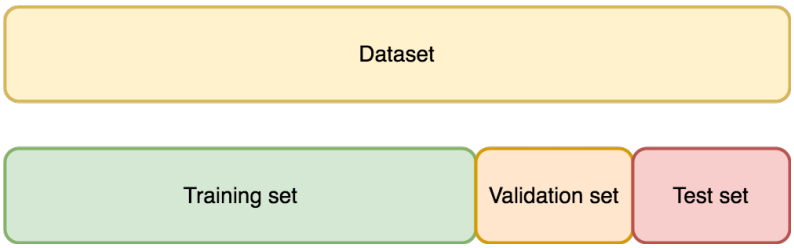


Figure 2.6: Model evaluation with held-out validation and test sets schema.

To solve the overfitting problem, another part of the dataset is held out to form the *validation set*. The training proceeds on the training set and the hyper-parameters adjustments is based on the results obtained from the validation set. When the model seems to be tuned properly, it is trained on the union of the training and validation sets with the best found hyper-parameters. Then the final evaluation is done on the test set.

The issues of this methodology are the substantial reduction of the number of training samples, the variance on all the three sets, and the generalization error on the test set because the results depend on the choice of the split [28].

Model evaluation with Cross-Validation and held-out test set

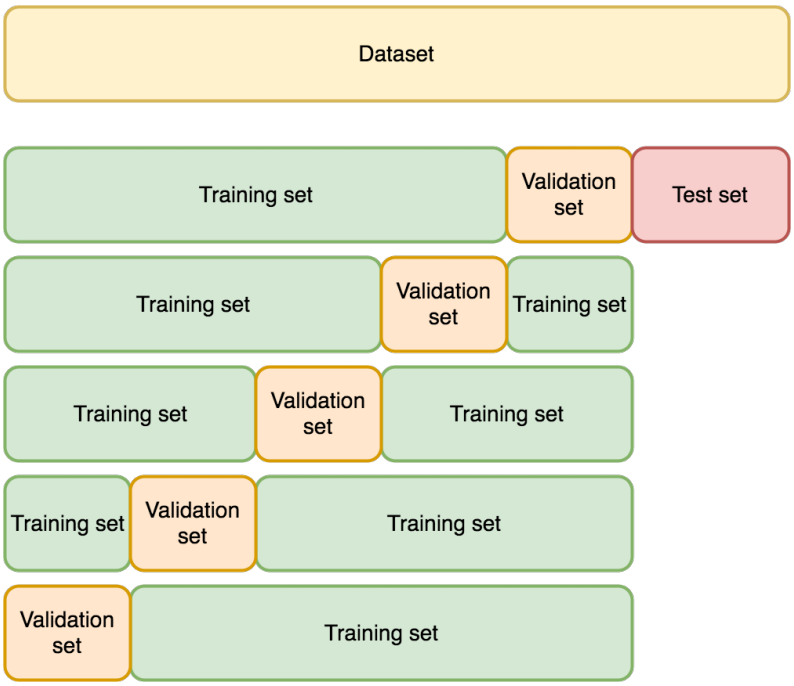


Figure 2.7: Model evaluation with a 5-fold CV and held-out test set schema.

A solution to the problems of the *held-out validation and test sets schema* is represented by the so called *Cross-Validation* (CV) procedure. It avoids to use the validation set of before. In fact, in the simplest approach of *k-fold CV*, the training set is split into k smaller sets. In turn, $k-1$ folds are used for training the estimator with certain hyper-parameters, and the remaining fold behaves as a validation set to compute a performance measurement (e. g. the accuracy). When the loop is terminated, the average of the scores obtained at each iteration can be considered a low variance estimation of the model performance with that particular choice of hyper-parameters. The higher is the number of folds, the more robust is the validation score from different training-test splits. Other alternatives to k-fold CV exist, such as *Repeated K-Fold*, *Leave One Out*, *Stratified k-fold*, etc. [28] and all of them follow in general the same principles.

Finally, various models are trained in this way with different combinations of hyper-parameters. Their prediction scores over the held-out test set are used to compare them: also this time the test set represent a source of generalization error.

The training and testing flow diagram is reported in Figure 2.8. The green block named "Cross-Validation" carries out the operations of Figure 2.7.

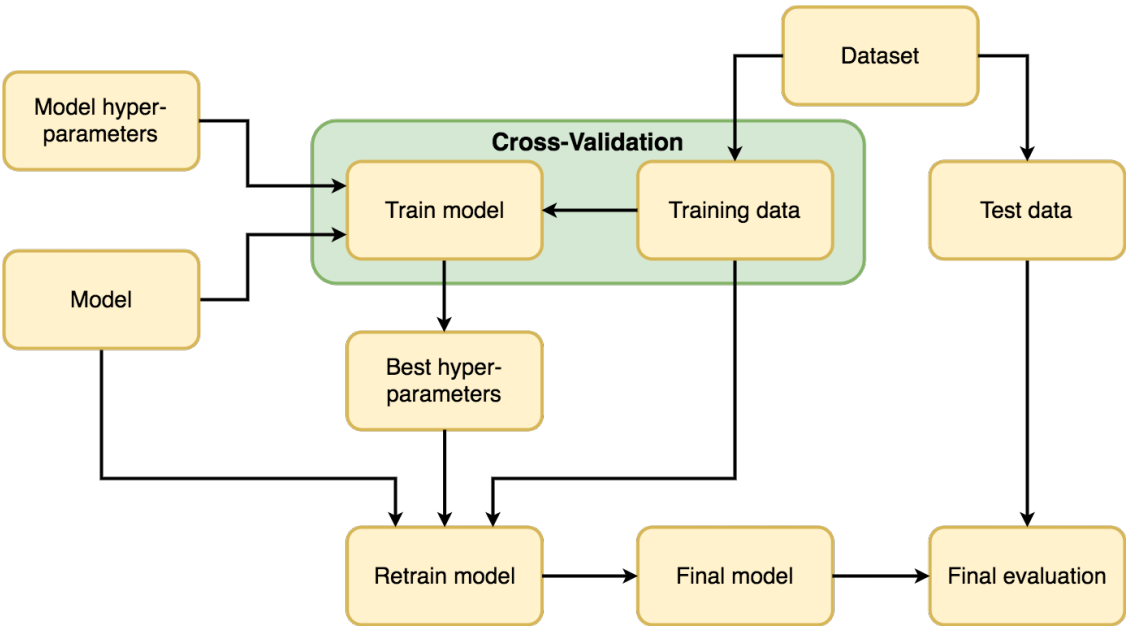


Figure 2.8: Flow diagram of the training and testing procedures in case of CV.

This method is probably the most famous one because offers many advantages with respect the other solutions. In particular the low variance on the validation scores, the prevention of the overfitting problem and the absence of an held-out validation set that would lead to a waste of useful data for the training phase. Since it is computationally expensive, the typical values for k (in case of k-fold CV) are 5 or 10 to have robust performance analysis and speed up implementation [37].

Model evaluation with Nested CV schema

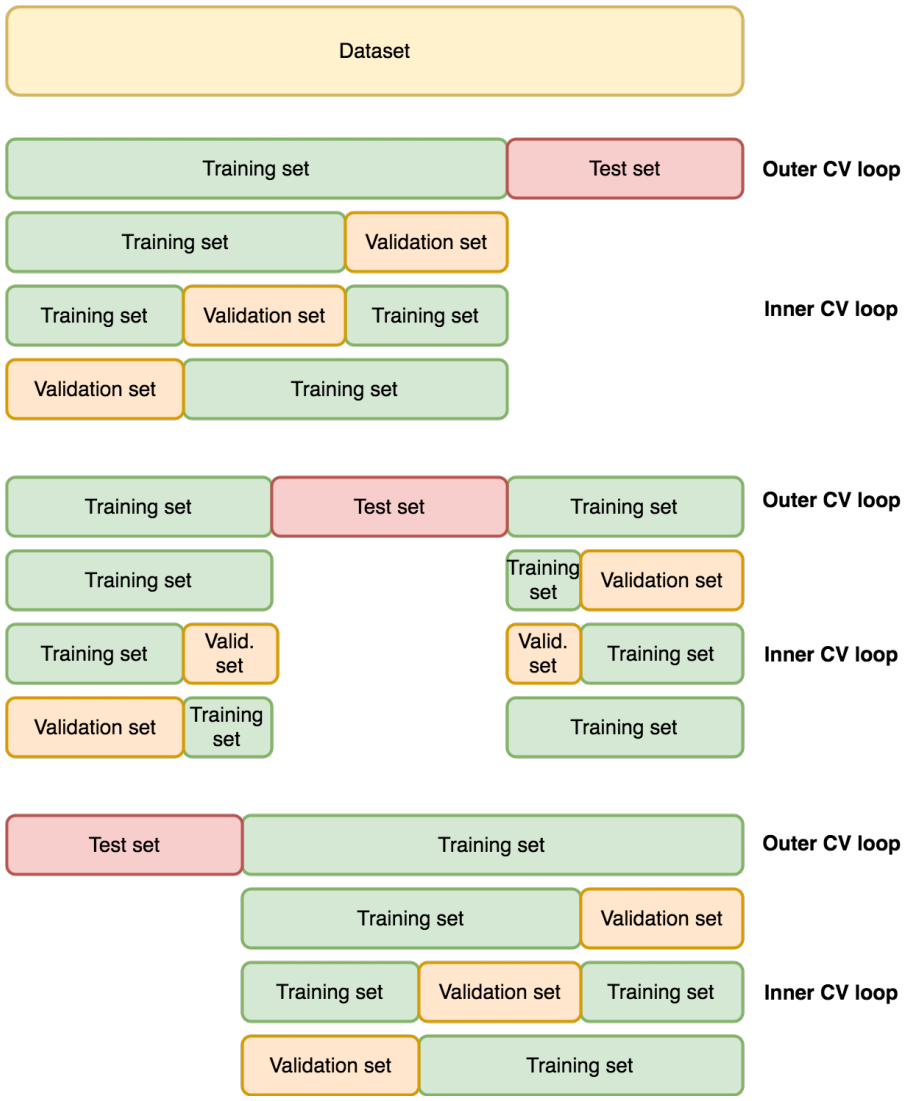


Figure 2.9: Model evaluation with Nested CV schema with 3-fold CV in both the inner and outer loops.

Nested CV is the solution to solve all the previous issues. It consists in two CV loops as can be seen in Figure 2.9 and 2.10. Consider again the basic k-fold approach for both loops, with r folds for the outer loop and s folds for the inner one (the method is also valid for different CV strategies). Starting with the outer loop, r-fold CV is applied on the entire dataset, splitting it in the usual training and test sets for r-times. Then, each training set is subjected to another s-fold CV as explained in the non-nested *Cross-Validation and held-out test set schema* of before: this is the inner loop. Therefore, there are $r \times s$ repetitions of non-nested CVs, which implies that the drawback of

this approach is the long training time.

The inner loop is used to find the best hyper-parameter by evaluating the model on the validation set splits and averaging the validation scores, as it happened before. After s iterations of a non-nested CV, the model is trained with the best found hyper-parameters on the entire training set for that particular training-test split. The outcome is a candidate model which is evaluated on the remaining test set fold. At the end of the inner loop the candidate model and its score are saved. In the next outer loop iteration a new training-test split is generated and the inner loop starts on the new training set. If the newly discovered candidate model is the same as before, its score is saved and averaged with the previous evaluations; otherwise this candidate model is normally saved together with its score.

This process is repeated for r outer iterations. Hence, in the worst case scenario there could be more than one different candidate models for a maximum of r. If this is the case, the analyst can follow at least two ways: she can either choose any of them, if the difference between their best hyper-parameters is small because all the candidate models exhibit robustness; or she can ensemble them in case of very different hyper-parameters.

In summary, the outer loop guarantees that the performance measurements of the best found models have a low generalization error on the test set because this time it is distributed on the entire dataset.

Due to the fact Nested CV is time-consuming, it is recommend for models with fast training and a small number of hyper-parameters to tweak, such as SVMs; instead it is not convenient to apply it for MLPs, in which non-nested CV is more suitable, even though non-nested CV could yield to overly-optimistic scores [28] and should be the common practice [38]. Typical values of r and s for the outer and inner CV loops are 10-10, 30-10, 30-30.

The training and testing flow diagram of the Nested CV schema is illustrated in Figure 2.10. The red block named "Nested Cross-Validation" carries out the operations of Figure 2.9, while the green block called "Cross-Validation" realizes the procedure of Figure 2.7.

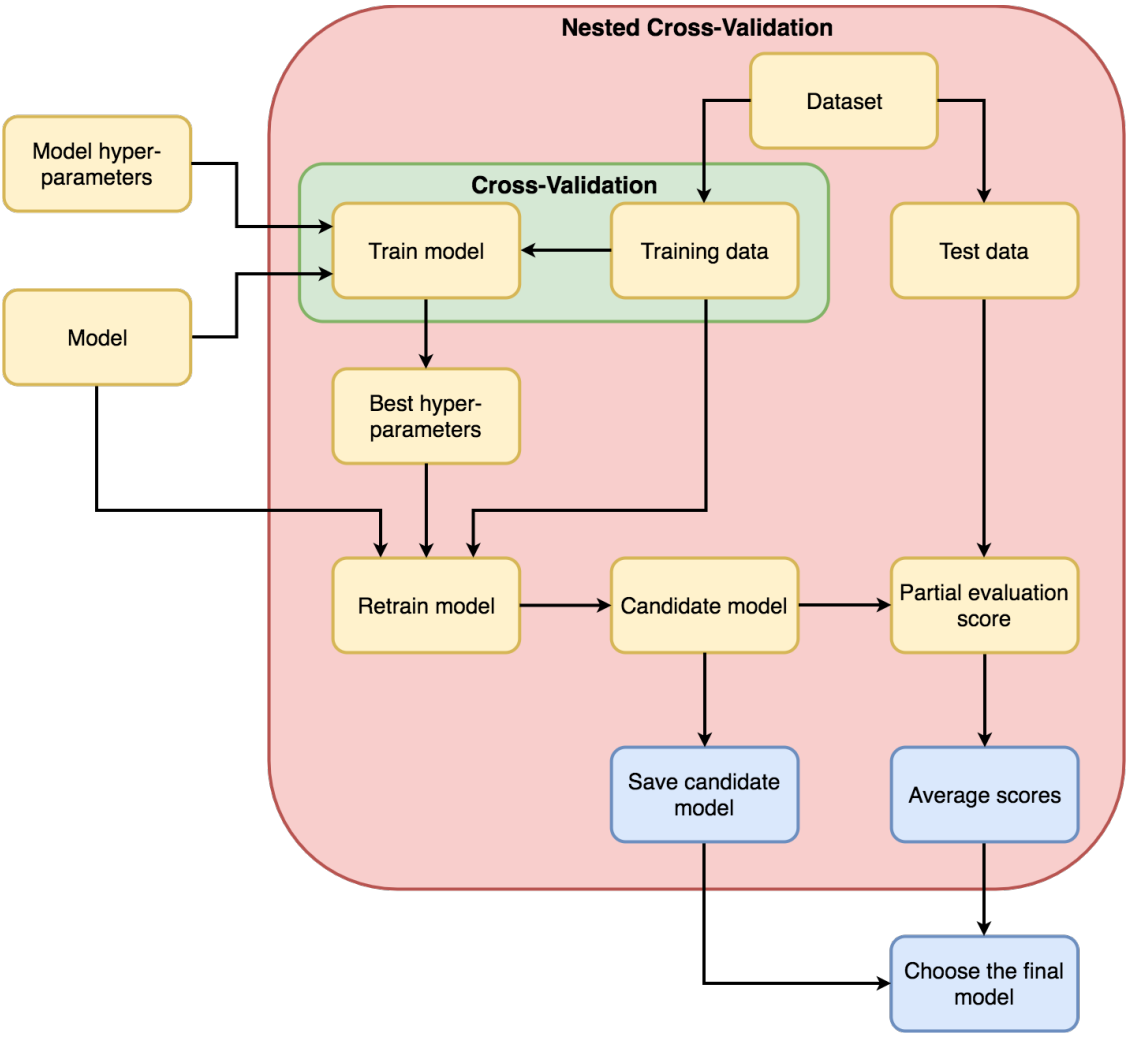


Figure 2.10: Flow diagram of the training and testing procedures in case of Nested CV.

It is important to underline that all these schemes have the goal to give an estimation of the models performance on new unseen data and so to select the optimal one. Subsequently, the best model has to be finalized, that is it has to be trained on the entire dataset with the corresponding best hyper-parameters, while the other models used for the performance estimation should be discarded. At this point the estimator is ready to be applied on-site to make new predictions.

2.2.5 Hyper-parameters tuning techniques

Hyper-parameters tuning is a crucial part for ML, and also the most tedious and difficult task. In fact, a good selection can boost classification accuracy, while a poorly configured model may perform no better than chance [39]. Generally speaking, the tuning consists in the following steps [28]:

1. select a ML model to train,
2. define a parameter space,
3. choose a method for searching or sampling candidates,
4. train the model at point 1) with the hyper-parameters selected at point 2) and 3) with one of the model evaluation schema of Paragraph 2.2.4,
5. evaluate the performance of each model with a score function, like those of Paragraph 2.2.6.

Depending on the problem to face and on the ML algorithm employed, the quest for the optimal hyper-parameters can be highly time consuming when the number of possible combinations is considerably high. For this reason there exist different ways to accomplish this task (point 3) of the previous list). They are listed in this section.

Manual-Search
This is the basic technique: when facing a new ML problem, the first thing that a developer does is to attempt to manually adjust some hyper-parameters to understand some general rules of thumb, such as their upper/lower bounds. Clearly, this solution is not applicable when dealing with more than a few hyper-parameters. The results of this approach can still be used in more sophisticated techniques to limit the grids of possible values of certain hyper-parameters to avoid wasting time in choices that have already proved to not give interesting results.

Grid-Search
This method is the most widely used and involves the creation of several grids of values, one per each hyper-parameter to tune. With the values inserted in these grids, a number of different models, equal to the number of all possible combinations of these hyper-parameters, is trained. At the end their scores (i.e. accuracy, in classification problems) are compared. That is why this brute-force approach is called *Grid-Search*. The first grids, sometimes called

"loose grids", usually contain values that are spaced linearly or logarithmically as power of 2 or 10 to explore the unknown solution space in a short time. Later the grids are refined towards the most promising values and a new Grid-Search is performed. The number of refinements is arbitrary.

Even if Grid-Search outperforms Manual-Search because the human intervention is reduced since many different trials can be executed automatically, it isn't the most effective strategy for expensive functions where the number of possible hyper-parameters combinations explodes, like neural networks.

Random-Search
As the name suggests, it samples the hyper-parameters from their grids with a random distribution (often uniform), instead of trying all possible combinations like the previous methodology. Despite of its triviality, in such cases this approach proved to be competitive with domain experts [39] in finding good settings. Its advantages over an exhaustive search are:

- the possibility to set a budget, that is the number of sampled candidates or sampling iterations,
- the fact that adding hyper-parameters that do not influence the performance does not decrease the efficiency of Random-Search [28].

Bayesian Optimization
Bayesian Optimization is one of the most efficient methods of function minimization for the evaluation of functions [39]. It is also known under the name *Sequential model-based optimization* (SMBO) when applied to ML. In short, it first builds a probability model, called *surrogate*, of the *objective function* (i.e. the target function to minimize); then uses it to predict the next set of hyper-parameters to try in the objective function; in the end updates the surrogate with the result of the objective function. It outperforms both Manual and Random-Search [40] because it chooses the most promising hyper-parameters based on the historical results that the previous hyper-parameters obtained in evaluating the target function. Therefore, the research efforts are concentrated on the most probable hyper-parameters. In this way the tuning time over a large set of hyper-parameters grids is dramatically reduced compared to the other methods.

To enter more in the details, the steps that are executed by the Bayesian Optimization algorithm are [41]:

1. build a surrogate probability model of the objective function. The common choices are Gaussian Processes, Random Forest Regressions, and Tree Parzen Estimators [42];
2. find the hyper-parameters that perform best on the surrogate using a *selection function*, such as *Expected Improvement*, *Entropy Search*, and *Knowledge Gradient* [42];
3. apply them to the true objective function and get the result (score);
4. update the history of the surrogate model with the new pair (score, hyper-parameters);
5. repeat point 2) and 4) until a maximum time or a maximum number of iterations is reached.

It is interesting to underline that the time spent to execute point 2) is negligible respect than evaluating hyper-parameters directly on the true objective function because the surrogate model is simpler to optimize than the real target function [41].

2.2.6 Metrics for performance evaluation

In order to appreciate the classification performance of a classifier on a given test set, there are plenty of ways. In this paragraph the most important and widely known are presented and adapted to the case of binary classification.

Accuracy score
It is defined as:

$$\begin{aligned} accuracy(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{correct_{samples}}{n_{samples}} = \\ &= \frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}} 1(\hat{y}_i = y_i) \end{aligned} \tag{2.42}$$

where $\mathbf{y}$ is the vector of the true labels, $\hat{\mathbf{y}}$ is the vector of the predicted labels, $correct_{samples}$ is the total number of samples classified correctly, and $n_{samples}$ is the number of samples used for the calculation or the length of the vectors $\mathbf{y}$ and $\hat{\mathbf{y}}$.

In binary classification *positive* and *negative* refer to the model prediction, and the terms *true* and *false* refer to the actual label [28]. So binary classifiers

are susceptible of two kinds of errors. A sample is defined False Positive (FP) when is predicted as positive, but in reality belongs to the negative class. In the opposite case, it is defined False Negative (FN). Instead, a data predicted correctly is defined True Positve (TP) or True Negative (TN) depending on its actual class. This implies that the accuracy score for a binary classifier can be written in another way:

$$accuracy = \frac{(TPs + TNs)}{(TPs + TNs) + (FPs + FNs)} \tag{2.43}$$

where TPs, TNs, FPs, and FNs are the number of samples predicted TP, TN, FP, and FN, respectively.

Precision and Recall scores

For binary classification, *precision* is the proportion of positive identifications that are actually correct. High precision relates to a low false positive rate. Its definition is the following:

$$precision = \frac{TPs}{(TPs + FPs)} \tag{2.44}$$

Instead *recall* is the proportion of actual positives that are identified correctly as positives. Sometimes it is called *Sensitivity*. High recall relates to a low false negative rate [28]. Its expression is:

$$recall = \frac{TPs}{(TPs + FNs)} \tag{2.45}$$

Both of them can be plotted in the *Precision-Recall curve* which shows their trade-off for different thresholds of the classifier. An ideal system should have high precision and high recall, which corresponds to a large area under the curve. This curve is useful to set the *operating point* of the model to the value that is required by the application. Indeed, by moving the decision threshold of the model, it is possible to maximize either precision or recall, remembering that they are antagonist. An example of Precision-Recall curve can be seen in Figure 3.18 of Paragraph 3.4.1.

Confusion Matrix

It is a practical tool to visualize the quality of the output of a classifier. The elements in the main diagonal are the number of correct predictions TNs and TPs, instead the elements outside the diagonal are the number of mistakes, i.e. FNs and FPs. The ideal condition is when these last contributions are

0. The definition of Confusion Matrix for binary classification is reported in Table 2.1.

True label	**TNs**	**FPs**
	FNs	**TPs**
	Predicted label	

Table 2.1: Definition of Confusion Matrix for binary classification.

In Figure 2.11 the Confusion Matrix of a SVM classifier trained on the Iris Dataset of Scikit-Learn is reported. This example shows a Confusion Matrix for a multi-class classification problem because the number of classes is greater than two, since there are three types of iris: setosa, versicolor and virginica. This model classifies all flowers well, except the versicolor, which is confused with a virginica six times.

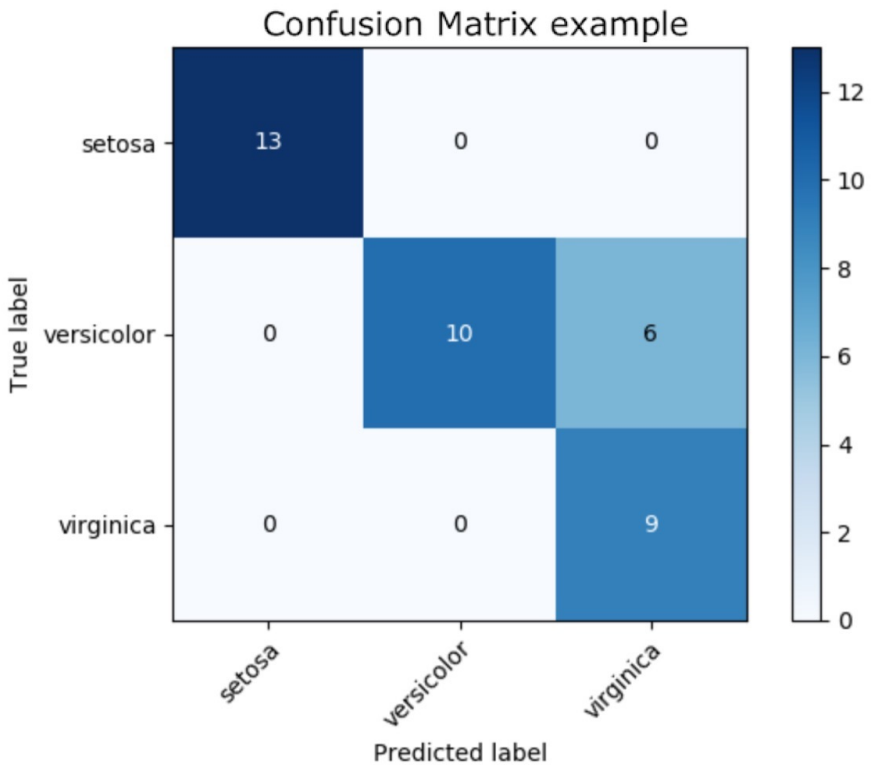


Figure 2.11: Confusion Matrix of a SVM classifier trained on the Iris Dataset of Scikit-Learn.

ROC curve

The name stands for *Receiver Operating Characteristic curve*. It shows the prediction ability of a classifier at all classification thresholds. This curve plots two parameters on the x and y axes: the *True Positive Rate* (TPR), which is synonym for recall, and the *False Positive Rate* (FPR), which is complementary to precision. Their expressions are:

$$TPR = \frac{TPs}{TPs + FNs} = recall \tag{2.46}$$

$$FPR = \frac{FPs}{FPs + TNs} = 1 - precision \tag{2.47}$$

An example of ROC curve is illustrated in Figure 2.12. By lowering the classification threshold, the model predicts more data as positive, increasing both TPs and FPs. An awful classifier has a curve close to the 45° bisector, while a good one has a curve that is near to the left and top borders. In other words, an operating point with coordinates (FPR, TPR) is index of an excellent model when it is close to the ideal (0, 1).

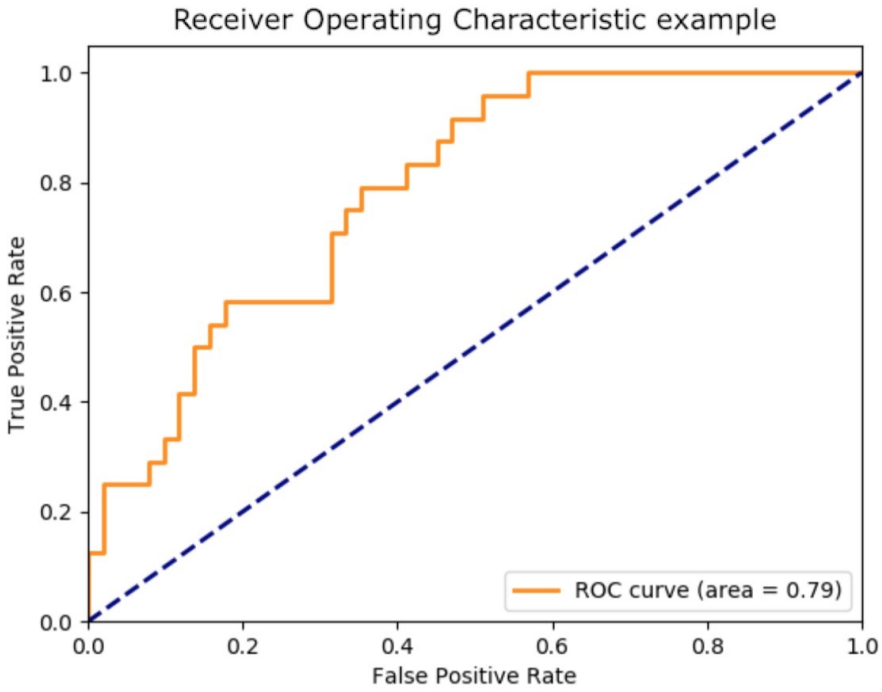


Figure 2.12: An example of a ROC curve.

Together with the ROC curve, another metric is employed: the *Area under*

the ROC curve (AUC). It is classification-threshold-invariant, that means it measures the quality of the predictions independently on what classification threshold is chosen [33]. For this reason it is often used as a summary of the model skill. The AUC ranges between 0.5 and 1.0, which are equivalent to random guessing and perfect classification for every decision threshold, respectively.

Chapter 3

Training and testing with Synthetic Data

This Chapter demonstrates how the Synthetic Dataset (the first of Paragraph 1.3) is generated and how the SVM and the MLP classifiers are trained and tested on it.

In the initial part, the simulation environment used to produce synthetic tomographic images is presented, with an explanation of the modeling of the antennas arch. Then, the procedure to create the Synthetic Dataset is reported in details. The preprocessing stage follows standardization to uniform the features and with PCA to reduce the high number of features. Different levels of PCA retained variance are employed to generate five different Synthetic Datasets from the initial one: 0.9, 0.925, 0.95, 0.975, 0.99. Each dataset is shuffled and then split in 80%-20% for training and testing.

So, as many SVMs as the number of Synthetic Datasets are trained with Grid-Search, first with a loose, then with finer grids: a 5-fold, 10-fold CV and 30-fold nested CV schemes are implemented. Then, three MLPs with 1, 2 and 3 hidden layers are trained with the best performing Synthetic Dataset found for the SVM: the one obtained by applying a retained variance of 0.95. For the MLPs the training approach involves a 5-fold CV scheme and the best hyper-parameters are searched with Bayesian Optimization. Finally, the classification performance of the best synthetic SVM and MLP models are evaluated on their Synthetic Test Sets and the results are compared throughout common metrics, such as Confusion Matrix and ROC Curve.

3.1 Synthetic Dataset creation

3.1.1 Simulated environment

Proceeding in order, the model used for the simulations of an hazelnut-cocoa cream jar is designed with *GiD*[1]. It is decided to consider a simple cylinder, whose base radius is 3.5 cm and its height is 8 cm, as reported in Figure 3.1.

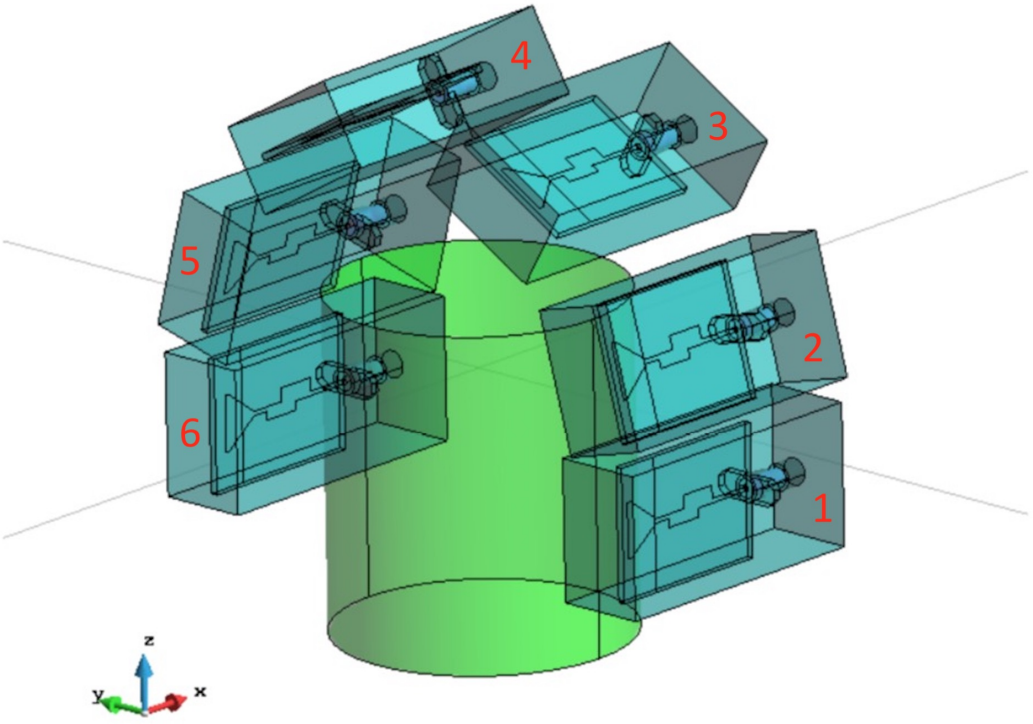


Figure 3.1: Simulated environment: antennas arch with hazelnut-cocoa cream underneath. Courtesy of Ricci M. [21].

The cylinder volume is entirely composed by chocolate cream, which has a dielectric constant of $\varepsilon_{cream} = 3$ @ 10 GHz[2]. It should be noticed that no

[1] `https://www.gidhome.com`

[2] Even if this is not the precise value (in fact the dielectric constant of the hazelnut-cocoa spread is $\varepsilon_{cream} = 2.86$ @ 10 GHz), at the time of the simulation it wasn't known yet. Only after the measurement reported in Figure 3.2b the value of 2.86 was discovered. Despite the error computed with this approximation is about 5%, it is below that of all the other existing approximations present in the simulated scenario, such as the shape of

jar is modeled because too thin to be relevant (the thickness of the real jar is in the order of 1 mm), always assuming it is made of a dielectric material to guarantee the MWI compatibility. The air medium fills the simulated environment around the cylinder with a dielectric constant equal to $\varepsilon_{air} = 1.00$ @ 10 GHz. The dielectric properties of the chocolate spread under test of Figure 3.2a are reported in Figure 3.3.

(a) **(b)**

Figure 3.2: Measurements of Nutkao's chocolate cream frequency characteristics with a VNA. Courtesy of Tobon V. J. [45].

The use of 10 GHz is the result of empirical measurements in the range $[1 \div 20]$ GHz with a VNA on the hazelnut-cocoa spread (Figure 3.2) to discover the dependencies of its dielectric constant (ε_r, Figure 3.3a), conductivity (σ, Figure 3.3b), penetration depth (PD, Figure 3.3c) and resolution ($\lambda/4$, Figure 3.3d) against frequency. The results say that PD is 3.5 cm and $\lambda/4$ is 4 mm at 10 GHz, and so this frequency allows a good trade-off among penetration depth, resolution, and costs/complexity of the electronic equipment for this application.

---

the jar, the position under the antenna arch, the level of the chocolate inside the jar, the discretization of the electric field in the volume, etc.

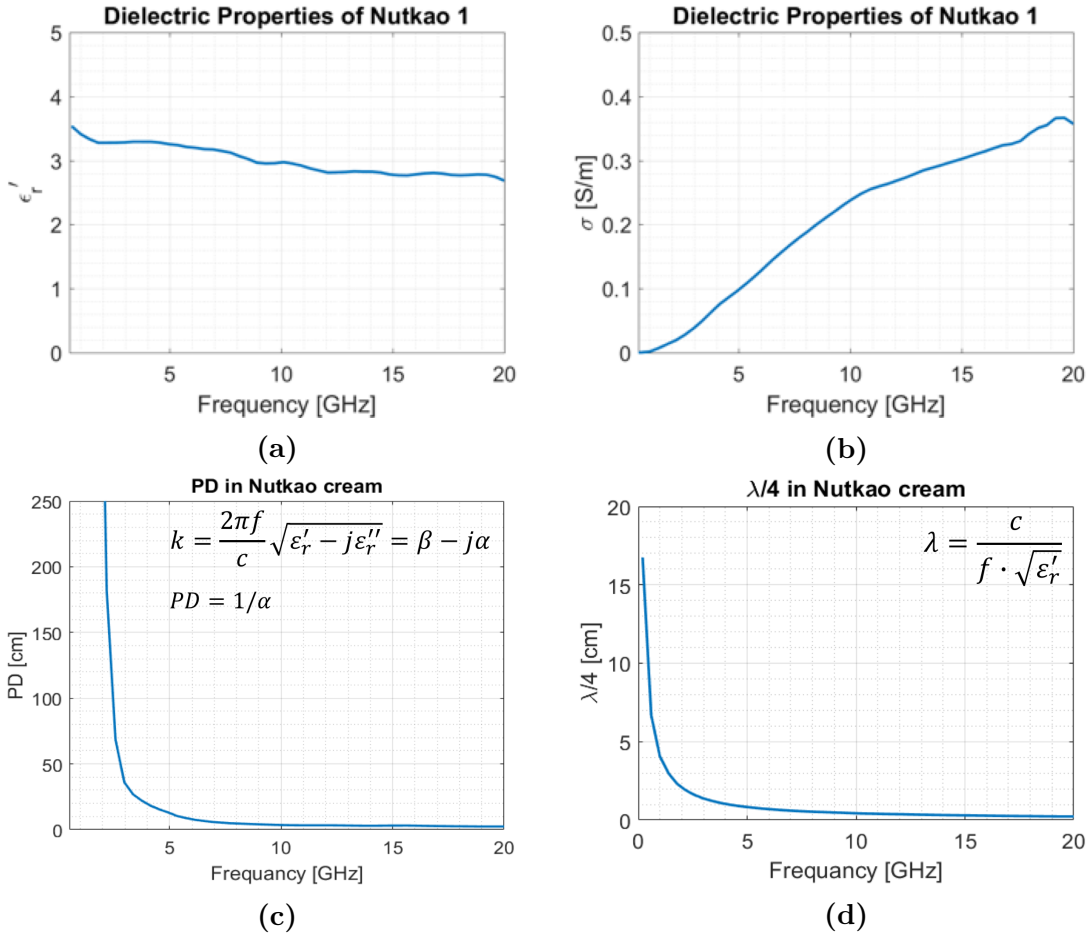


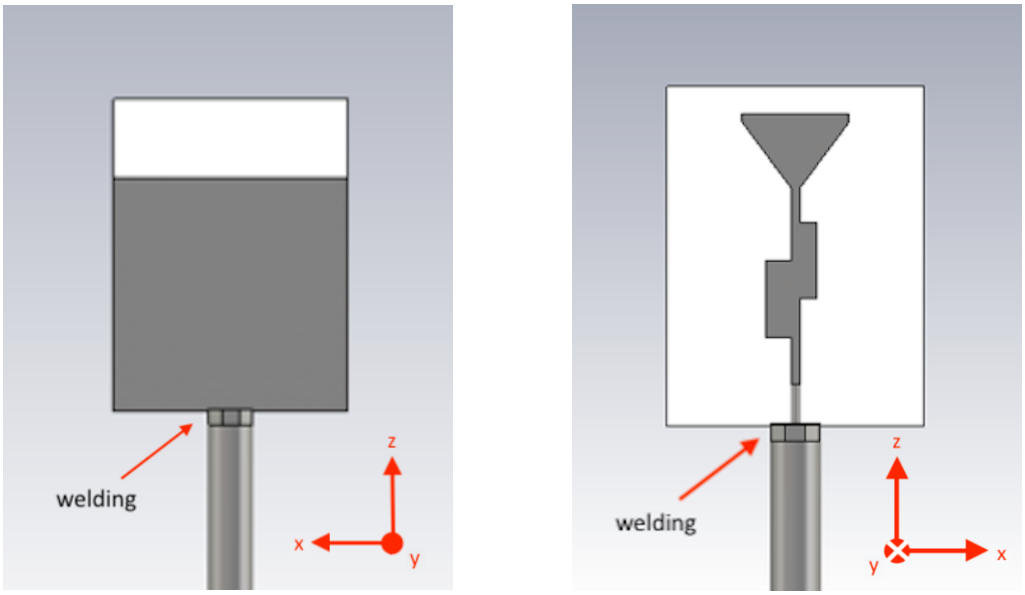
Figure 3.3: Results of the measurements of the Nutkao's chocolate cream frequency characteristics. a) Real part of the dielectric constant. b) Conductivity. c) Propagation depth. d) Resolution. Courtesy of Tobon V. J. [45].

3.1.2 Antennas arch electromagnetic characteristics

The antennas arch is a multi-antenna system and is visible in Figure 3.1. It is made of six triangular aperture PCB antennas which are shown in Figure 3.4. The antennas are disposed in half circle: antennas 3 and 4 are rotated by 15°, 5 and 6 by 60°, antennas 1 and 6 are diametrically opposed but positioned 1 cm below the ideal half circle. The arch is placed around the object to inspect to exploit multiple views and to be compatible with a future in-line adoption in food industries.

The triangular aperture PCB antenna properties are described in Figure 3.5 and 3.6, and the key elements are summarized here:

- this kind of antenna is low-cost and easy to build up;
- the simulated antenna radiates best at roughly 9.6 GHz;
- the radiation pattern in far-field is not highly directive because covers almost all the front of the antenna uniformly. The main lobe is at 233° (with respect the positive side of the x-axis and rotating counter-clockwise) with a magnitude of 3.77 dBi, an angular width @ 3 dB of 78.9° and a side lobe level of -0.5 dB.



(a) **(b)**

Figure 3.4: Back a) and front b) of a triangular aperture PCB antenna. The sides of the PCB are 4 cm x 3 cm. The grey color represent copper. The substrate is FR4. In the back a ground plane is designed. Courtesy of Ricci M. [44].

3.1.3 Synthetic Dataset creation procedure

Remembering the Equation 2.1 and 2.2 in Paragraph 2.1, the calculus of $\mathcal{L}$ requires the knowledge of the "background" electric field. For this reason, after having designed the simulation environment in *GiD*, the 3D model is meshed in tetrahedrons with a 1.2 mm grid. Then the resulting model is imported in a custom software which implements the Finite Element Method (FEM) to solve the necessary differential equations and boundary conditions numerically and find the "background" electric fields $\boldsymbol{E}_b(\boldsymbol{r}_p, \boldsymbol{r}_m)$ and $\boldsymbol{E}_b(\boldsymbol{r}_m, \boldsymbol{r}_q)$ for

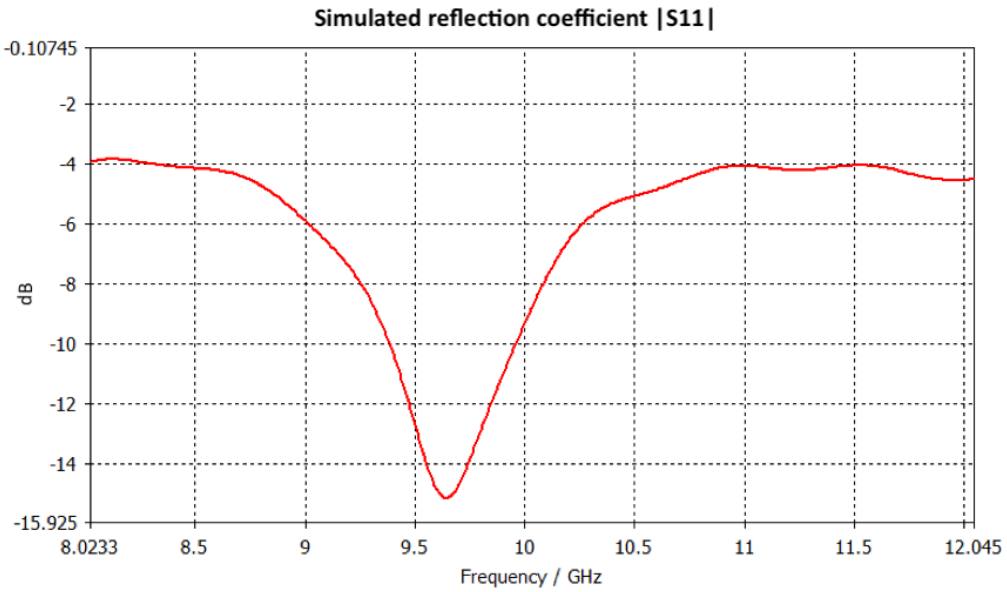


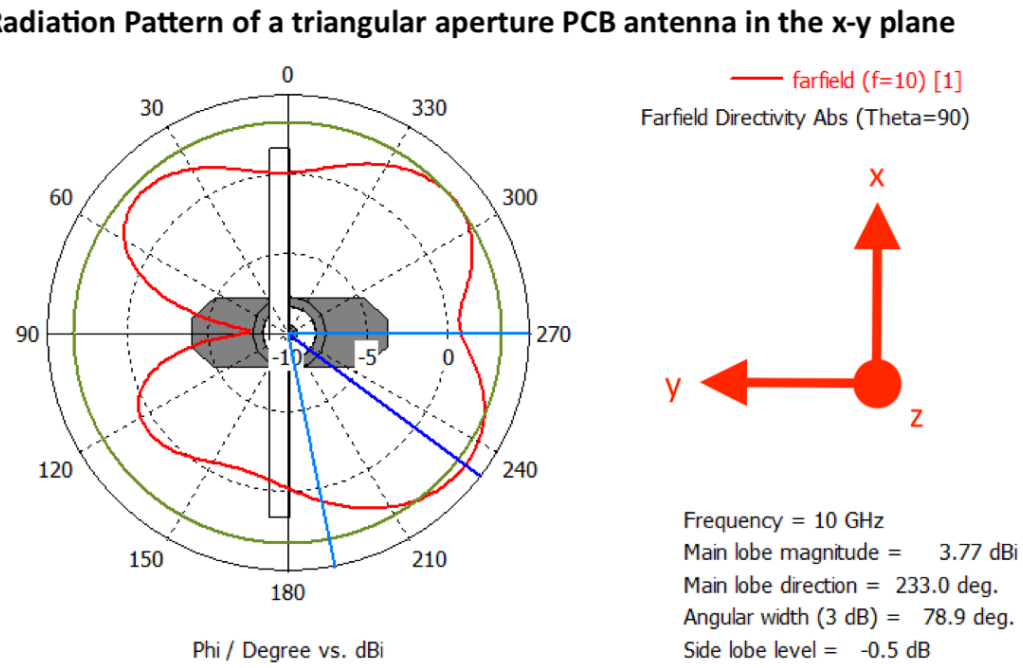
Figure 3.5: Simulated reflection coefficient of the triangular aperture PCB antenna.

all combination of antennas p and q and for all the positions $\boldsymbol{r}_m$ inside the volume of the jar.

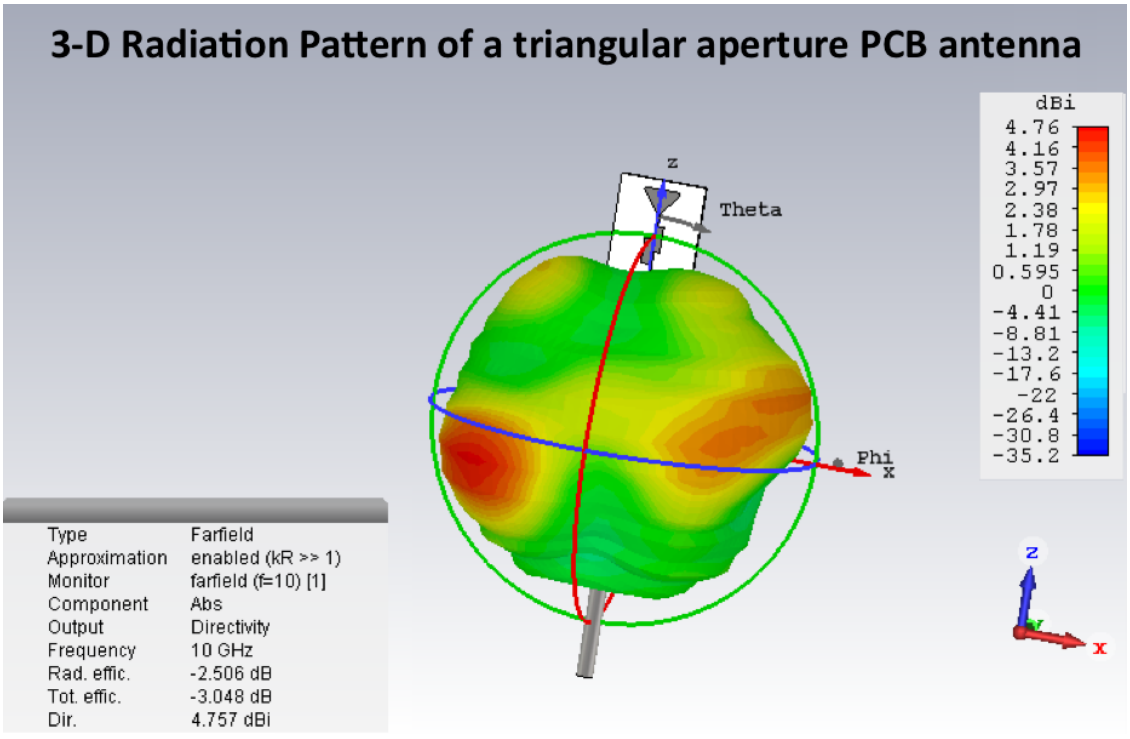
The next step involves the TSVD of the $\mathcal{L}$ operator, whose outputs are the singular values $\boldsymbol{S} = \{\sigma_n\}$ and the associated vectors $\boldsymbol{U} = \{u_n\}$ and $\boldsymbol{V} = \{v_n\}$. The total number of singular values is 36. It represents all the possible combinations of transmitting and receiving antennas (one transmitting and one receiving). The truncation is done after the first $m = 15$ largest singular values because they give the highest contribution of information. This means that 15 over 36 are the necessary couples of transmitting and receiving antennas to be switched to reconstruct the tomographic image of the jar. Hence, 15 is the number of elements of the upper triangular part of the Scattering-matrix of the network composed by the jar, the antennas arch and the air in between, without considering the self scattering s-parameters, i.e. the main diagonal. The magnitude of the singular values of $\mathcal{L}$ are reported in Figure 3.7.

A *dielectric contrast* with respect to background material ($contrast_{imp}$) is imposed to a selected group of tetrahedrons in order to simulate the presence of a contaminant inside the cylindrical volume. This contrast is given by Equation 3.1:

$$contrast_{imp} = \frac{\varepsilon_{obj} - \varepsilon_{bg}}{\varepsilon_{bg}} + AWGN \tag{3.1}$$



(a) 2-D triangular aperture PCB antenna radiation pattern in the x-y plane @ 10 GHz (the antenna z-axis is exiting the paper).



(b) 3D triangular aperture PCB antenna radiation pattern @ 10 GHz.

Figure 3.6: 2D and 3D triangular aperture PCB antenna radiation patterns @ 10 GHz. Courtesy of Ricci M. [44].

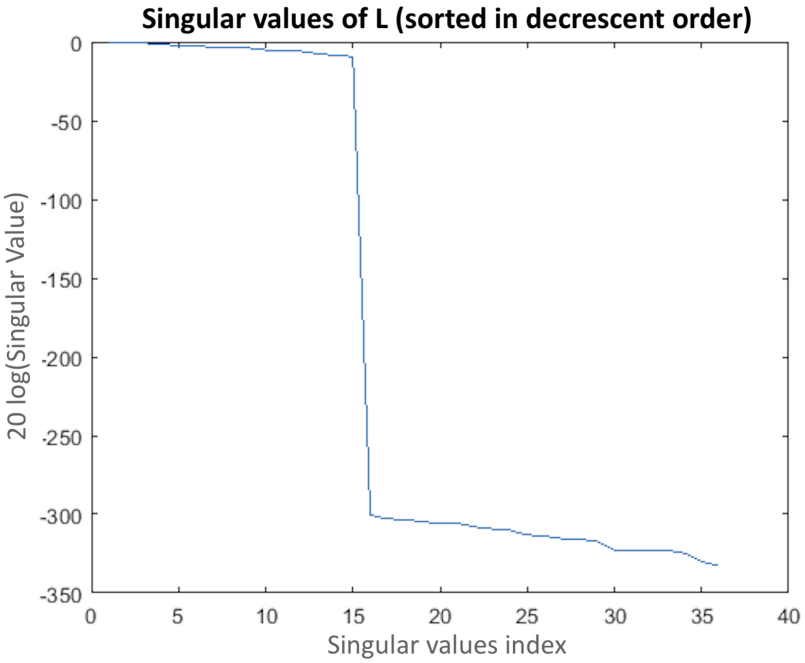


Figure 3.7: Magnitute of the singualar values of $\mathcal{L}$.

where ε_{obj} is the dielectric constant of the intrusion, ε_{bg} is the dielectric constant of the background material (the chocolate spread); $AWGN$ (Additive White Gaussian Noise) is added to the imposed contrast to better simulate a real-case scenario.

The position in the volume and the size of the contaminant is set by imposing the wanted contrast (Equation 3.1) to a cluster of tetrahedrons, whose center coincides to the desired location of the wanted intrusion and whose sphere radius corresponds its dimension. The tetrahedrons that are not inside that sphere radius and are cut by the sphere surface are not included in the cluster and their contrast is only noise. This can be easily proven by forcing ε_{obj} to ε_{bg} in Equation 3.1. At this point, every tetrahedron in the cylindrical volume has an imposed dielectric contrast.

Later, the contrast of the entire volume is projected onto the TSVD domain of the $\mathcal{L}$ operator by performing Equation 3.2, and the tomographic image is retrieved with Equation 3.3:

$$coef_{imp} = (contrast_{imp} \bullet \mathbf{V})^* \tag{3.2}$$

$$project = \sum_{i=1}^{n} coef_{imp_i} \cdot v_i \tag{3.3}$$

where $coef_{imp}$ stands for "imposed coefficients", $*$ is the complex conjugate symbol, $\bullet$ is a dot product, v_i is the corresponding eigenvector belonging to $\boldsymbol{V} = \{v_n\}$ which is obtained from the TSVD of $\mathcal{L}$, $project$ is the reconstructed tomographic image, and n is the number of the necessary couples of transmitting and receiving antennas to be switched to reconstruct the tomographic image of the jar. This time n is 21 instead of 15 because the reconstruction of the tomographic image in simulation proved to be better with 21 singular values. So, 21 is the number of elements of the upper triangular part of the Scattering-matrix of the network composed by the jar, the antennas arch and the air in between, including the main diagonal.

It is worth to notice the similarity between these two last formulas and Equation 2.4 of Paragraph 2.1, even if the approaches to generate a synthetic data and a real data are different. As a matter of fact, the first case imposes a known dielectric contrast change to a small volume in the jar, the cluster. Then, it projects the contrast onto the TSVD of the linear operator $\mathcal{L}$ of the simulated reference scenario. The result is a tomographic image with the presence of a contaminant in the position where the dielectric contrast change was applied. Instead, the normal procedure starts from the variations of the scattering parameters measured from the network, composed by the jar under inspection, the antennas arch and the air in between, and from the golden case, solves the linear inverse problems with a TSVD and arrives to extract the dielectric contrast change in the jar, which would be the same tomographic image if a real intrusion was placed in the same position of the synthetic approach. Therefore, the advantage of the first methodology is the control of the position, of the dielectric constant and of the size of the intrusion in the volume with the goal of generating the same tomographic image that would be obtained with the standard procedure for the same kind of contaminant. Moreover, the first strategy permits to decide arbitrary values of dielectric constant for the intrusion.

As anticipated in Paragraph 1.3, the Synthetic Dataset is created to validate the idea of applying ML to the foreign body detection problem in the hazelnut-cocoa cream jars with MWI. It is composed by 1800 synthetic data of tomographic images: 900 contaminated and 900 uncontaminated chocolate spread jars. This means that the dataset is balanced. Throughout the thesis, the two classes will be referred as "contaminated" class and "free" class. Examples of two samples belonging to the contaminated and free classes was already reported in Figure 1.1 where they are plotted in 3D tomographic images.

Let's describe the Synthetic Dataset generation procedure. Every sample is generated with a *Matlab* script which behaves a bit differently if the sample has to be contaminated or not. For a contaminated sample, the script:

1. **chooses the dimension of the intrusion**: the sphere radius of the cluster of tetrahedrons is picked from the range $[2 \div 10]$ mm with an uniform distribution;
2. **selects the position of the intrusion**: the 3 spacial coordinates of the center of the cluster of tetrahedrons are generated with an uniform distribution;
3. **checks if the intrusion is confined inside the cylinder volume**: the sum of each coordinate with the sphere radius must not exit the volume. When it happens, steps 1 and 2 are repeated;
4. **prepares a noise term**: the noise value is chosen to be in the range $[-0.1 \div 0.1]$ with a Gaussian distribution with mean 0 and standard deviation 0.025;
5. **imposes a $contrast_{imp}$ to the tetrahedrons inside the sphere and assigns only the noise term to all the others**;
6. **computes Equation 3.2 and Equation 3.3**;
7. **saves the tomographic image (*project*) into a textual file**: since it is an array of complex numbers, it is flattened in a vector of real numbers where the real and the imaginary parts of each entry are placed one after the other.

Instead, about the production of uncontaminated/free samples, remembering Equation 3.1, the only term that is required is the noise. Thus, for an uncontaminated sample, the script executes steps 4, 5, 6 and 7.

For replicability of the Synthetic Dataset generation process, the *Matlab* random generator is initialized every 100 samples. The value for the initialization is even and starts from 0, for the uncontaminated samples, and is odd and begins with 1, for the contaminated ones. To clarify, the first 100 uncontaminated samples are produced with seed = 0, the second 100 with seed = 2, etc.; the first 100 contaminated samples are produced with seed = 1, the second 100 with seed = 3, and so on.

3.2 Synthetic Dataset preprocessing

Dataset preprocessing follows dataset generation. First of all, the entire Synthetic Dataset is shuffled and then split in two parts, a training set that consists of 80% of the data and a Synthetic Test Set composed by the remaining 20%. It is important to underline the number of samples belonging to the "contaminated" class and the "free" class are equal in both training and test sets. In this way training and test sets are still balanced.

The next step consists in standardizing the training data and applying the same standardization to the test set. The motivations are already explained in 2.2.3.

Later, PCA is implemented to reduce the number of features. Indeed, the length of one sample is enormous: 3 563 930 features. In these situations, training a classifier leads to poor classification performances [43] because of the Curse of Dimensionality, as discussed in 2.2.3. The solutions to this problem are mainly two: either increase the number of training samples or reduce the number of features. As the former implies a substantial growth of the disk storage because a sample occupies around 30 MB, the latter is the way out. Hence, five PCAs with different levels of retained variance, equal to 0.9, 0.925, 0.95, 0.975 and 0.99, are executed to generate five new datasets used to train the ML algorithms in Paragraph 3.3. This choice is due to the fact there is no a priori knowledge on the problem and it is not possible to predict which level of retention is more suitable.

Synthetic Dataset	**Dataset Size (100%)**	**Retained Variance**	**# of Extracted Features**	**Synthetic Training Set Size (80%)**	**Synthetic Test Set Size (20%)**
1	1800	0.90	12	1440	360
2	1800	0.925	15	1440	360
3	1800	0.95	18	1440	360
4	1800	0.975	25	1440	360
5	1800	0.99	32	1440	360

Table 3.1: The five different datasets with the corresponding number of extracted features according to the level of PCA retained variance that is applied.

As PCA can be also used to visualize a dataset, in Figure 3.8 the plot of the Synthetic Dataset projected onto the 2 principal components extracted with PCA is reported. The goal of this graph is only illustrative because the

total retained variance is 35.377%, that means the level of information that is transferred to the 2 principal components is poor. This implies that if a classifier is trained with this new dataset, its performances won't likely be good because of the lack of information. However, this 2-D representation is interesting because it lets to visualize part of the dataset that is going to be fed to the classifiers: in red the contaminated samples, in green the uncontaminated/free ones.

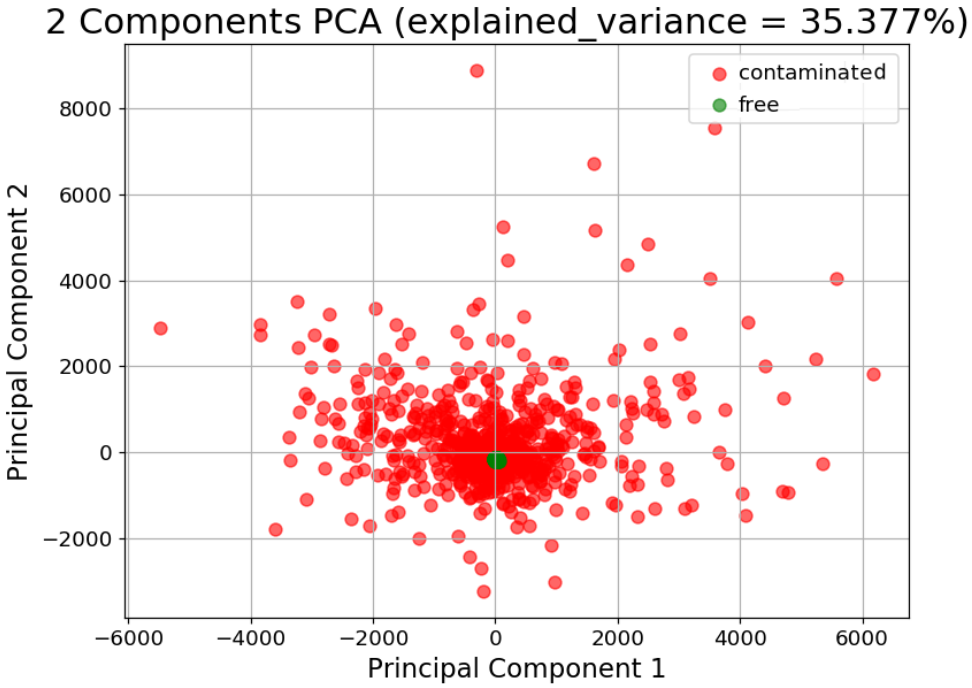


Figure 3.8: Plot of the Synthetic Dataset projected onto the 2 principal components extracted with PCA. In red the contaminated samples, in green the uncontaminated/free ones.

3.3 Training procedures and synthetic candidate models

The training procedure is not unique for the two classifiers taken into account. This section explains both training strategies and provides the best model hyper-parameters for both the classifiers.

The software tools and libraries used for dealing with ML are listed in Table 3.2.

Tool/Library Name	**Formal Library Name**	**Version**	**Purpose**
Python		3.7.4	Programming language
Anaconda Navigator		1.9.7	GUI to lunch Jupyter Notebook and manage libraries installation
Jupyter Notebook	jupyter-client jupyter-console jupyter-core	5.3.3 6.0.0 4.5.0	Code Editor
Scikit-Learn	scikit-learn scikit-image	0.21.3 0.15.0	Machine Learning Python library
Keras	keras	2.2.4	Neural Networks Python library
TensorFlow	tensorflow	1.14.0	Keras dependency
Hyperopt	hyperopt	0.1.2	Bayesian Optimization Library
NumPy	numpy	1.17.2	Math library
Matplotlib	matplotlib	3.1.0	Library for figures
Pandas	pandas	0.25.1	Data structures and data analysis library
SciPy	scipy	1.3.1	Math library

Table 3.2: List of software tools and libraries used for dealing with ML.

3.3.1 SVMs training and synthetic candidate models

Starting with the SVM, the procedure is inspired by [29]. The following steps are repeated for each of the five Synthetic Training Sets of Table 3.1 in Paragraph 3.2:

1. consider the Radial Basis Function (RBF) kernel: it is the kernel suggested in [29]. In addition, at the very beginning a Linear kernel was employed, but it led to poor classification performances ($< 90\%$ of accuracy) and was discarded immediately;

2. use cross-validation (CV) and Grid-Search to find the best hyper-parameters C and γ that maximize the CV accuracy:

   (a) to overcome the long training time required by Grid-Search, start

with a loose grid and gradually make a finer grid towards the best hyper-parameters found with the previous grid;

(b) carry out different k-fold CV strategies hand in hand with the grid refinements: from 5-fold CV, passing for 10-fold CV, to 30-fold nested CV (30-fold in the inner loop, 30-fold in the outer);

3. use the best parameter C and γ of the last finest grid to train the classifier on the whole training set;

4. test the trained classifier with the best parameter C and γ on the test set and extract the performance evaluation metrics necessary to compare it with other solutions.

The most relevant training results of the SVM on the five Synthetic Datasets are reported in five separate tables: Table 3.3 for Synthetic Dataset 1, Table 3.4 for Synthetic Dataset 2, Table 3.5 for Synthetic Dataset 3, Table 3.6 for Synthetic Dataset 4, and Table 3.7 for Synthetic Dataset 5. The columns *C Grid* and γ *Grid* contain the start and stop exponents of a logarithmic scale with base 10 and the number of points used to divide the exponents interval. For example:

- "[-1, 2] 4p" means the exponent interval has 4 points from -1 to 2 (included) which are -1, 0, 1 and 2, and so the grid is composed by: 10^{-1}, 10^{0}, 10^{1}, 10^{2};

- "[-1, 2] 8p" means the exponent interval has 8 points from -1 to 2 (included) which are -1, -0.571, -0.143, 0.286, 0.714, 1.143, 1.571, 2, and so the grid is composed by: 10^{-1}, $10^{-0.571}$, $10^{-0.143}$, $10^{0.286}$, $10^{0.714}$, $10^{1.143}$, $10^{1.571}$, 10^{2}.

The column *Best* (C, γ) includes the couples of hyper-parameters that give the highest cross-validation score for that grid, which is in the next column named *Best CV Accuracy* (%). In the case two or more couples have the same CV accuracy, the one with the lower C is selected, as explained in Paragraph 2.2.1.

Grid Type	**# Folds**	C **Grid**	γ **Grid**	**Best** (C, γ)	**Best CV Training Acc. (%)**
Loose 1	5	[-5, 10] 16p	[-15, 3] 19p	(1.0E+06, 1.0E-07)	98.750
				(1.0E+02, 1.0E-05)	98.750
				(1.0E+00 1.0E-04)	98.750
				(1.0E+04, 1.0E-06)	98.750
Loose 2	5	[-3, 7] 11p	[-8, -3] 12p	(2.7E-02, 2.2E-04)	98.958
Fine	**10**	[-3, 1] 5p	[-5, -3] 9p	**(2.2E-02, 2.0E-04)**	**98.958**
				(6.0E-02, 2.6E-04)	98.958
				(1.7E-01, 3.4E-04)	98.958

Table 3.3: SVM training results for Synthetic Dataset 1 (PCA retained variance = 0.90). The best (C, γ) and its CV accuracy are in bold.

Grid Type	**# Folds**	C **Grid**	γ **Grid**	**Best** (C, γ)	**Best CV Training Acc. (%)**
Loose 1	5	[-5, 10] 16p	[-15, 3] 19p	(1.0E+00, 1.0E-04)	98.819
				(1.0E+06, 1.0E-07)	98.819
				(1.0E-02, 1.0E-04)	98.819
				(1.0E+04, 1.0E-06)	98.819
				(1.0E+02, 1.0E-05)	98.819
Loose 2	5	[-3, 7] 11p	[-8, -3] 12p	(8.0E-02, 2.2E-04)	99.097
Fine	**10**	[-3, 1] 5p	[-5, -3] 9p	**(7.7E-03, 1.5E-04)**	**99.097**
				(2.2E-02, 2.0E-04)	99.097

Table 3.4: SVM training results for Synthetic Dataset 2 (PCA retained variance = 0.925). The best (C, γ) and its best CV training accuracy are in bold.

Grid Type	**# Folds**	C **Grid**	γ **Grid**	**Best** (C, γ)	**Best CV Training Acc. (%)**
Loose 1	5	[-5, 10] 16p	[-15, 3] 19p	(1.0E-01, 1.0E-04)	98.958
Loose 2	5	[-3, 7] 22p	[-8, -3] 24p	(8.0E-02, 2.2E-04)	99.097
				(9.0E-03, 1.4E-04)	99.097
Fine	**10**	[-3, 1] 10p	[-5, -3] 6p	**(7.7E-03, 1.5E-04)**	**99.167**
				(1.7E-01, 2.6E-04)	99.167
				(4.6E-01, 3.4E-04)	99.167
				(2.2E-02, 2.0E-04)	99.167

Table 3.5: SVM training results for Synthetic Dataset 3 (PCA retained variance = 0.95). The best (C, γ) and its best CV training accuracy are in bold.

Grid Type	**# Folds**	C **Grid**	γ **Grid**	**Best** (C, γ)	**Best CV Training Acc. (%)**
Loose 1	5	[-5, 10] 16p	[-15, 3] 19p	(1.0E-02, 1.0E-04)	99.028
Loose 2	5	[-3, 7] 22p	[-8, -3] 24p	(8.0E-02, 1.4E-04)	99.028
				(2.7E-02, 1.4E-04)	99.028
				(2.4E-01, 1.4E-04)	99.028
				(7.2E-01, 1.4E-04)	99.028
				(9.0E-03, 8.2E-05)	99.028
Fine	**10**	[-3, 1] 10p	[-5, -3] 6p	**(7.2E-03, 8.7E-05)**	**99.028**
				(7.2E-03, 1.1E-04)	99.028
				(1.9E-02, 8.7E-05)	99.028
				(1.9E-02, 1.1E-04)	99.028
				(1.9E-02, 1.5E-04)	99.028
				(5.2E-02, 1.1E-04)	99.028
				(5.2E-02, 1.5E-04)	99.028

Table 3.6: SVM training results for Synthetic Dataset 4 (PCA retained variance = 0.975). The best (C, γ) and its best CV training accuracy are in bold.

Grid Type	**# Folds**	**C Grid**	**γ Grid**	**Best (C, γ)**	**Best CV Training Acc. (%)**
Loose 1	5	[-5, 10] 16p	[-15, 3] 19p	(1.0E-02, 1.0E-04)	99.028
				(1.0E-01, 1.0E-04)	99.028
				(1.0E+00, 1.0E-04)	99.028
Loose 2	5	[-3, 7] 22p	[-8, -3] 24p	(2.4E-01 1.4E-04)	99.028
				(8.0E-02, 1.4E-04)	99.028
				(9.0E-03, 8.2E-05)	99.028
				(2.7E-02, 1.4E-04)	99.028
				(2.7E-02, 8.2E-05)	99.028
				(7.2E-01 , 1.4E-04)	99.028
				(2.2E+00, 1.4E-04)	99.028
Fine	**10**	[-3, 0] 8	[-5, -3] 6p	**(7.2E-03, 8.7E-05)**	**99.028**
				(7.2E-0, 6.7E-05)	99.028
				(1.9E-02, 8.7E-05)	99.028
				(1.9E-02, 1.1E-04)	99.028
				(5.2E-02, 8.7E-05)	99.028
				(5.2E-02, 1.1E-04)	99.028
				(5.2E-02, 1.5E-04)	99.028

Table 3.7: SVM training results for Synthetic Dataset 5 (PCA retained variance = 0.99). The best (C, γ) and its best CV training accuracy are in bold.

Instead, in Table 3.8 the SVM training results on the five Synthetic Datasets are summarized. Every row corresponds to the best performing SVM on each Synthetic Dataset. The best SVM is in bold with a 10-fold CV accuracy of 99.167%. These final SVMs will be tested in Paragraph 3.4.1.

Synthetic Dataset	**Retained Variance**	**# Folds**	**Best (C, γ)**	**Best CV Training Acc. (%)**
1	0.90	10	(2.2E-02, 2.0E-04)	98.958
2	0.925	10	(7.7E-03, 1.5E-04)	99.097
3	**0.95**	**10**	**(7.7E-03, 1.5E-04)**	**99.167**
4	0.975	10	(7.2E-03, 8.7E-05)	99.028
5	0.99	10	(7.2E-03, 8.7E-05)	99.028

Table 3.8: Summary of the SVM training results on the five Synthetic Datasets. Every row corresponds to the best performing SVM relatively to its Synthetic Datasets. The best SVM is in bold.

For completeness, the outcomes of the training procedure just for the case of Synthetic Dataset 3, the best dataset, are given in the following.

The first three plots represent the Validation Curves vs. C and vs. γ for the case of the 10 folds, while the other hyper-parameter is fixed. They are useful to select the hyper-parameters C and γ that give the highest CV accuracy. For example, the best C and γ reported in Table 3.5 for 10 folds are derived from this graph because they produce the peaks of accuracy, which corresponds to 99.167%.

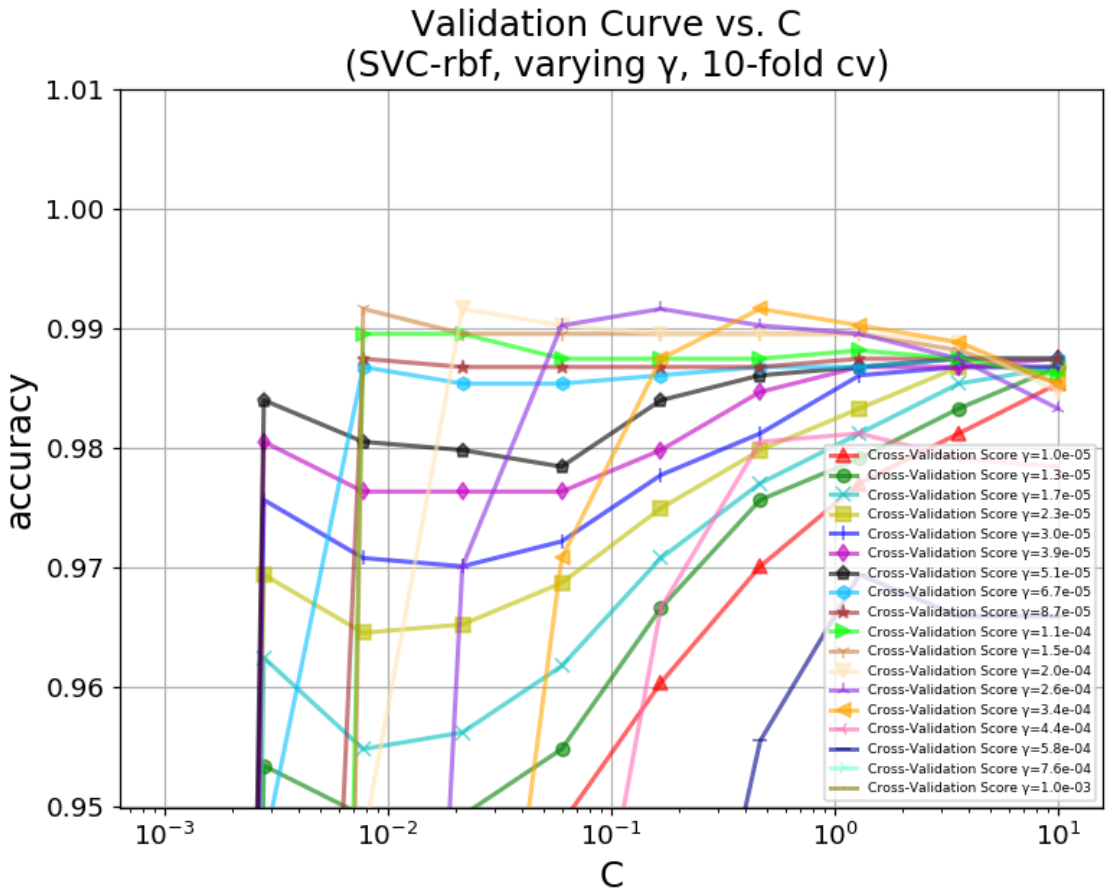


Figure 3.9: Validation Curve vs. C for the SVMs trained on Synthetic Dataset 3 with the fine grid reported in Table 3.5.

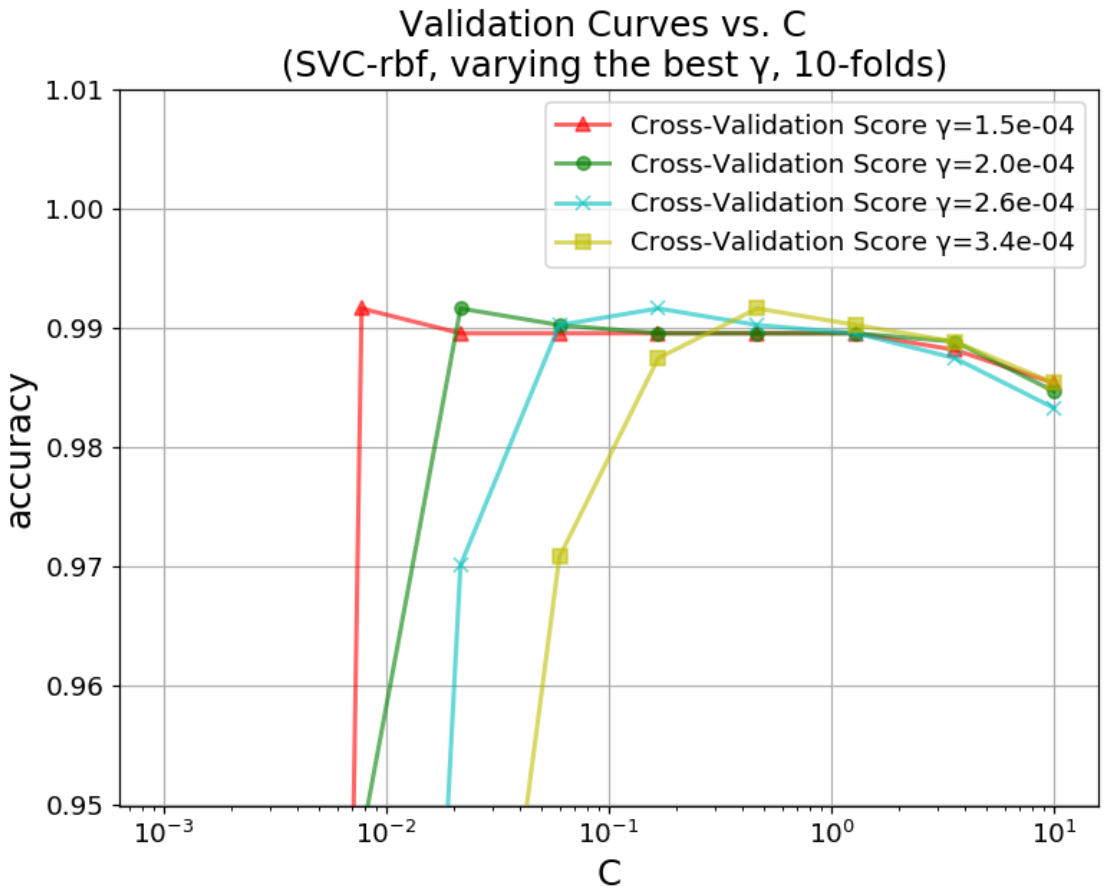


Figure 3.10: Validation Curve vs. C for the SVMs trained on Synthetic Dataset 3 with the fine grid reported in Table 3.5. The four curves that give the maximum 10-fold CV accuracy of 99.167% are kept with respect to Figure 3.9 for a better visualization.

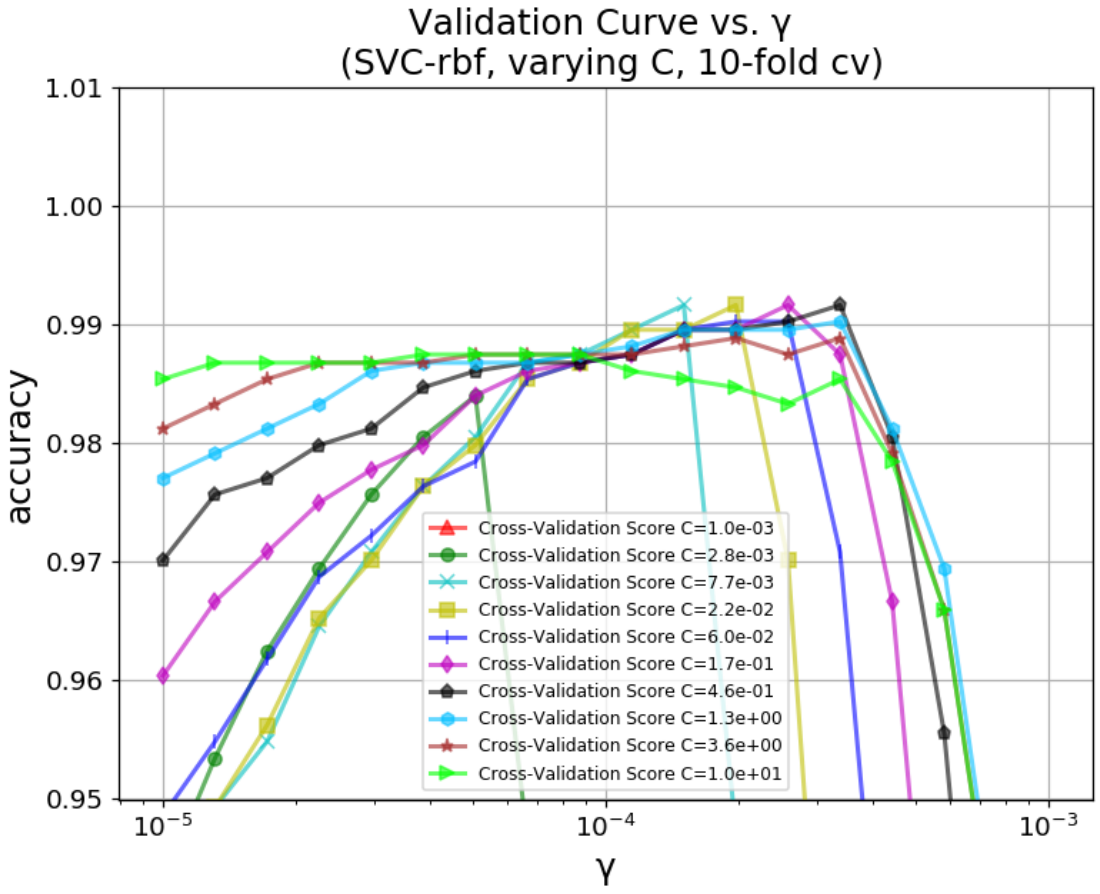


Figure 3.11: Validation Curve vs. γ for the SVMs trained on Synthetic Dataset 3 with the fine grid reported in Table 3.5.

The next four plots, in Figure 3.12, represent the "classical" Validation Curves of the four best SVMs for the case of the 10 folds. "classical" means that both training and validation accuracy are reported vs. one of the hyper-parameters, usually C, while the other, γ, is fixed. In this way, this kind of graphs are able to tell if a given choice of hyper-parameters (C, γ) influences the overfitting or underfitting of the classifier. What can be seen is:

- none of these four cases is overfitting for the couple (C, γ) marked with a red cross. In fact, that point is always positioned where there is maximum validation accuracy;
- in all of them, the training and validation accuracy are both high and with low discrepancy in the points marked by the red crosses. This indicates the classifiers are working well, neither overfitting, nor underfitting,

and that the Grid-Search method found the best values.

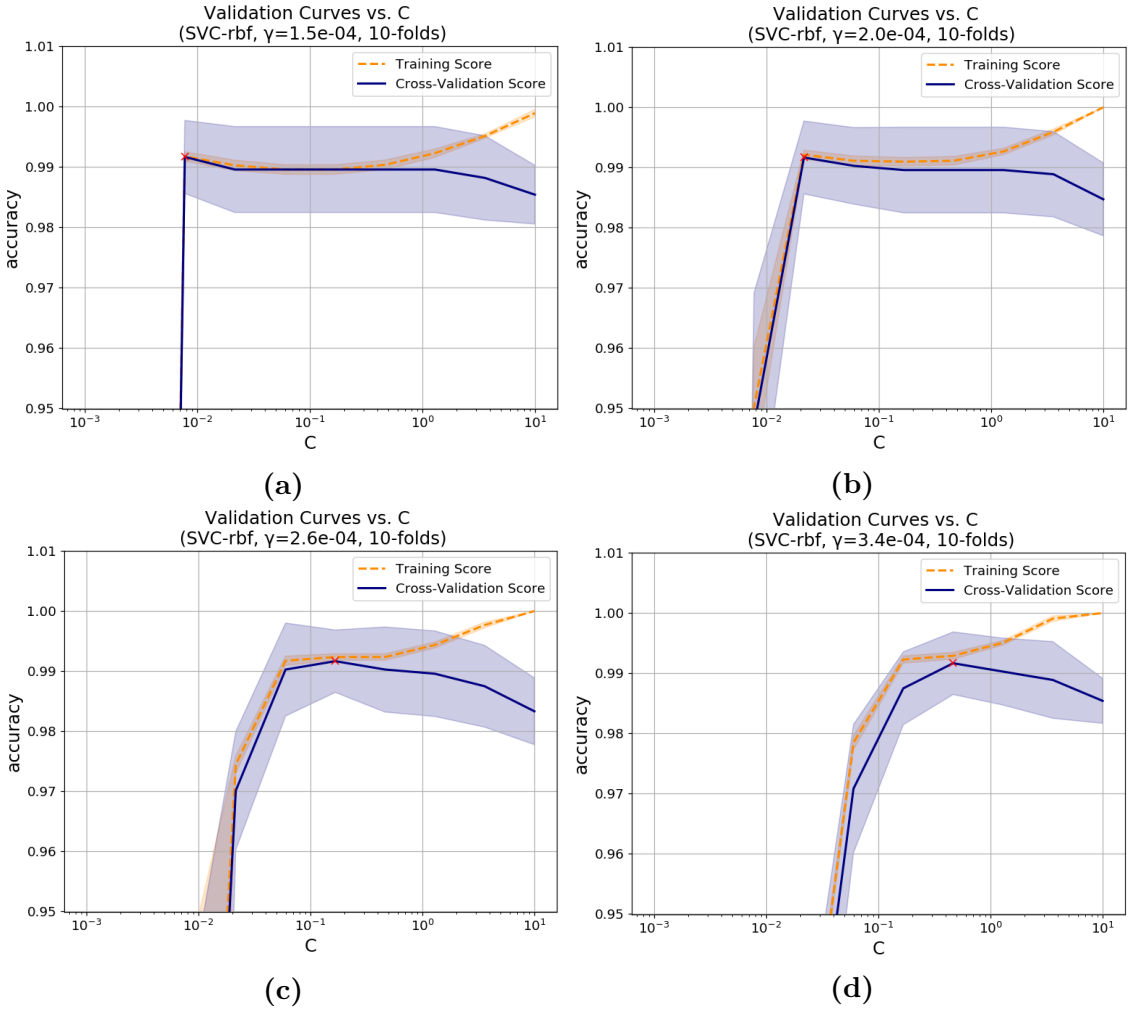


Figure 3.12: "Classical" Validation Curves vs. C of the four SVMs that give the maximum 10-fold CV accuracy of 99.167%, trained on Synthetic Dataset 3 with the fine grid reported in Table 3.5.

The last four plots, in Figure 3.13, represent the Learning Curves of the same four best SVMs for the case of the 10 folds. Again both training and validation accuracies are reported. This time the x-axis represents the training set size. So, this graphs are important to understand if more training data could be useful to improve the CV accuracy and if the classifier suffers for a *bias error* or a *variance error*. The *bias* measures the accuracy of the match between the classifier and the problem, while the *variance* measures the precision of the match [20]. In this case both are low because the training

curve touches the maximum accuracy and the validation curve follows the training curve with no discrepancy. On the other hand there is no need to increase the dataset size because the four models have already saturated to their maximum values.

The lines are the interpolation of the 10-fold CV accuracy points which are calculated for an increasing training set size. Instead the shadowed region around the lines is formed by the standard deviations of the accuracies calculated among the 10 folds.

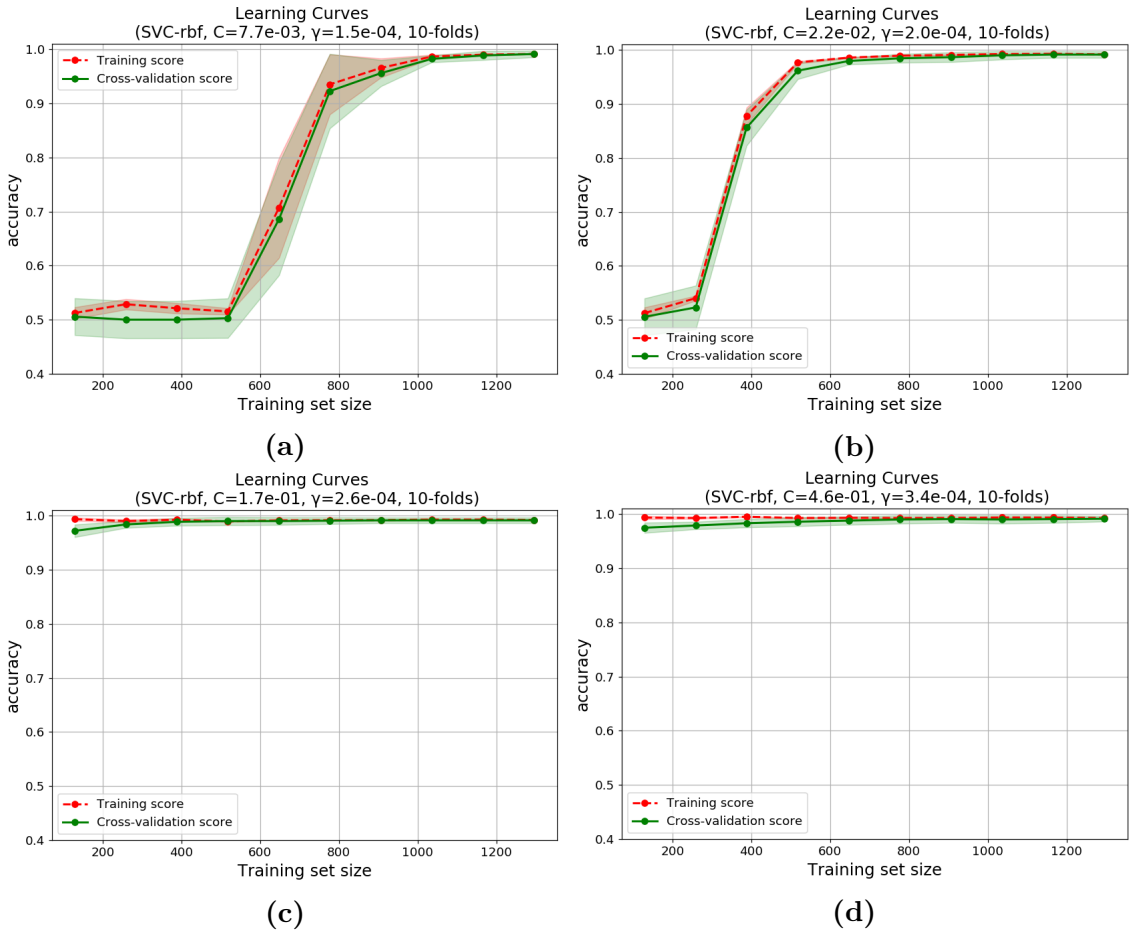


Figure 3.13: Learning Curves of the four SVMs that give the maximum 10-fold CV accuracy of 99.167%, trained on Synthetic Dataset 3 with the fine grid reported in Table 3.5.

3.3.2 MLPs training and synthetic candidate models

The training procedure for the MLP classifier is completely different from the previous one. Because of the high number of possible hyper-parameters to tune, the Grid-Search approach is prohibitive for its huge computational time. Hence, a different method is employed known as Bayesian Optimization, already discussed in Paragraph 2.2.5. The goal is to search for the hyper-parameters that minimize the validation loss. Since the training time is reduced but still considerable, the MLP is trained only on Synthetic Dataset 3, the one which provided the best results for the SVM.

Three types of MLPs are implemented, with 1, 2 and 3 hidden layers. All of them have an input layer equal to the number of features, which is 18 in the case of Synthetic Dataset 3 (with PCA reained variance = 0.95), and an output layer consisting in one sigmoid neuron, because the MLP is dealing with a binary classification problem. The choice of three hidden layers comes from [20], as already discussed in Paragraph 2.2.2.

The subsequent steps explain the procedure that is followed for training the three MLPs classifiers:

1. define a loose grid and then a fine grid for each tunable hyper-parameter with the possible values that the Bayesian Optimization algorithm can pick for the training (Table 3.9). The fine grid is chosen in the direction of the most performing hyper-parameters found with the loose grid;

2. define constant hyper-parameters which are not tunable, that is they remain always the same for the entire training phase;

3. configure early-stopping to monitor the validation loss, with 10 epochs of patience and a triggering condition of 0.002[3];

4. use 5-fold CV and Bayesian Optimization to find the best hyper-parameters that minimize the validation loss. Nested CV is not performed because of the already high training time of a MLP;

5. use the best hyper-parameters of the last finest grid to train brandly new MLPs on the 75% of the initial Synthetic Training Set 3 until the epoch

[3]early-stopping is a technique in charge of stopping the current training when a certain condition is met, even if the maximum number of epochs is not reached yet, to avoid overfitting and saving time. In this case, when the validation loss increases or doesn't improve (diminishes) of at least 0.002 for 10 subsequent epochs, the current training ends.

that provides the lowest validation loss. The validation set is created from the 25% of the initial training set;

6. test the trained classifier with the best hyper-parameters on the test set and extract the performance evaluation metrics necessary to compare it with other solutions.

Regarding the tuned hyper-parameters, which are explained in Paragraph 2.2.2, some of them are kept constant:

- Optimizer: Adam, to avoid to tweak the learning rate;
- Loss Function: Binary Cross-Entropy;
- Weight Regularization: L2, active when Weight Regularization Parameter is greater than 0;
- Batch-size: 50;
- maximum number of Epochs: 1000.

while others are tuned:

- number of Units per hidden layer,
- Activation Functions: Relu and Selu, because they proved to give the lower training and validation losses respect to tanh, softmax, sigmoid and softsign, for this benchmark case: kernel initializer=He Normal, weight regularization=0, dropout=0, optimizer=Adam, loss function=Binary Cross-Entropy, batch-size=50, with 10 units in each hidden layer for the three kinds of MLPs with 1 (Figure 3.14), 2 (Figure 3.15) and 3 (Figure 3.16) hidden layers;
- Kernel Initializer (or Weight Initializer): He Normal and Lecun Normal, because the first is usually adopted together with Relu and the second with Selu [46], and because of their properties described in 2.2.2;
- Weight Regularization Parameter;
- Dropout Rate: it is decided to use a value lower than 0.6 of dropped units per hidden layer.

Grid Type	# Folds	Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Activation Grid	Weight Initializer Grid	Weight Regularization Parameter Grid	Dropout Rate Grid
loose	5	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]			[Relu, Selu]	[He Normal, Lecun Normal]	[0.0, 0.0001, 0.001, 0.01, 0.1]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
fine	5	[from 16 to 256 with step 8]			[Relu, Selu]	[Lecun Normal]	[0.0]	[0.35, 0.4, 0.45, 0.5, 0.55]
loose	5	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]		[Relu, Selu]	[He Normal, Lecun Normal]	[0.0, 0.0001, 0.001, 0.01, 0.1]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
fine	5	[32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]		[Relu, Selu]	[Lecun Normal]	[0.0]	[0.4, 0.45, 0.5, 0.55]
loose	5	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[Relu, Selu]	[He Normal, Lecun Normal]	[0.0, 0.0001, 0.001, 0.01, 0.1]	[0.4, 0.45, 0.5, 0.55]
fine	5	[32, 64, 128, 256, 512, 1024]	[4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[Relu, Selu]	[Lecun Normal]	[0.0]	[0.4, 0.5, 0.55]

Table 3.9: Grids for the three types of MLP (1, 2 and 3 hidden layers) for Synthetic Dataset 3. The fine grids come from the results of the loose grids, reported in Table 3.10, 3.11 and 3.12.

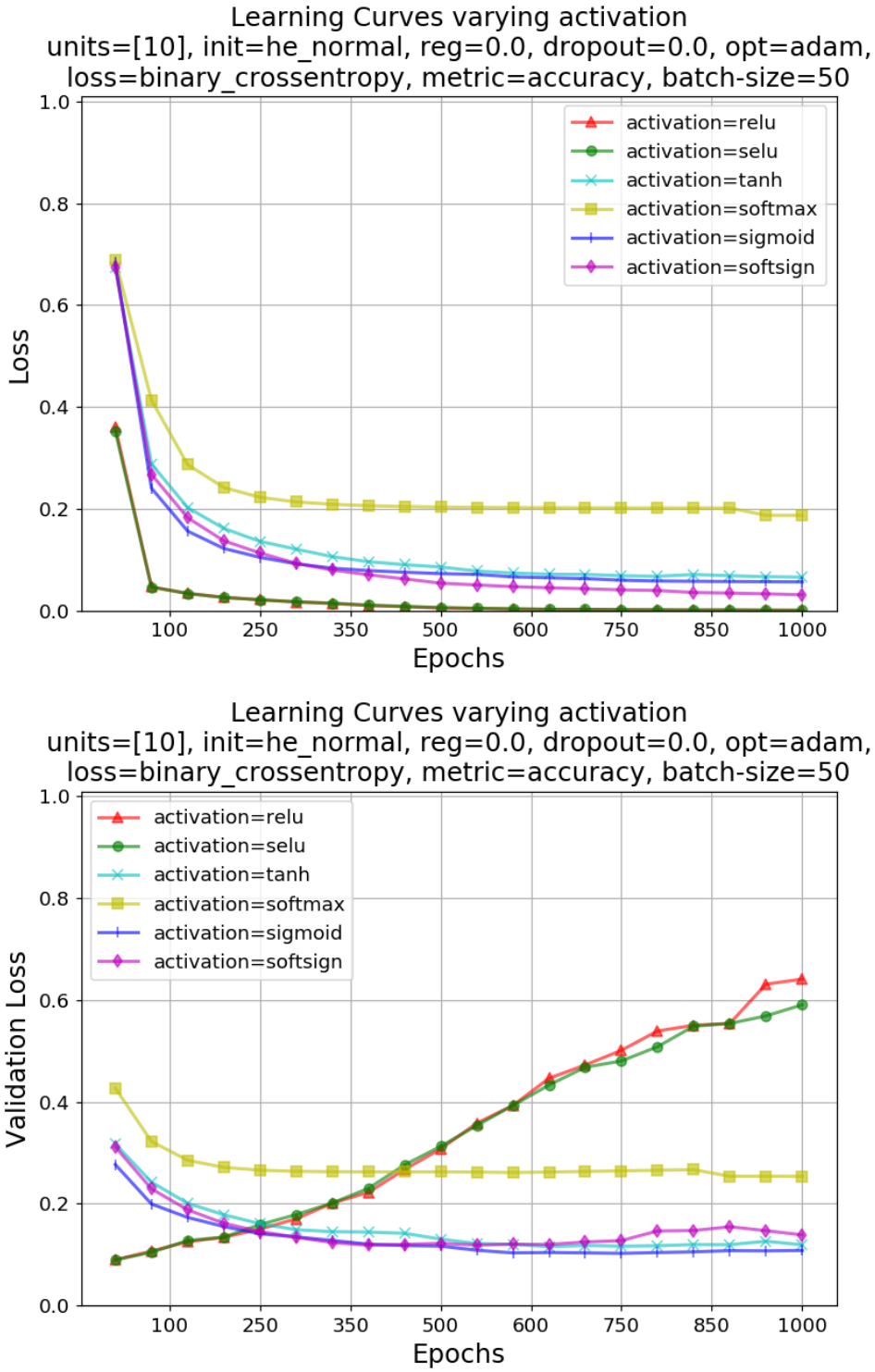


Figure 3.14: 1 hidden layer MLP benchmark for selecting the Activation Functions to use in the Bayesian Optimization grid for Synthetic Dataset 3.

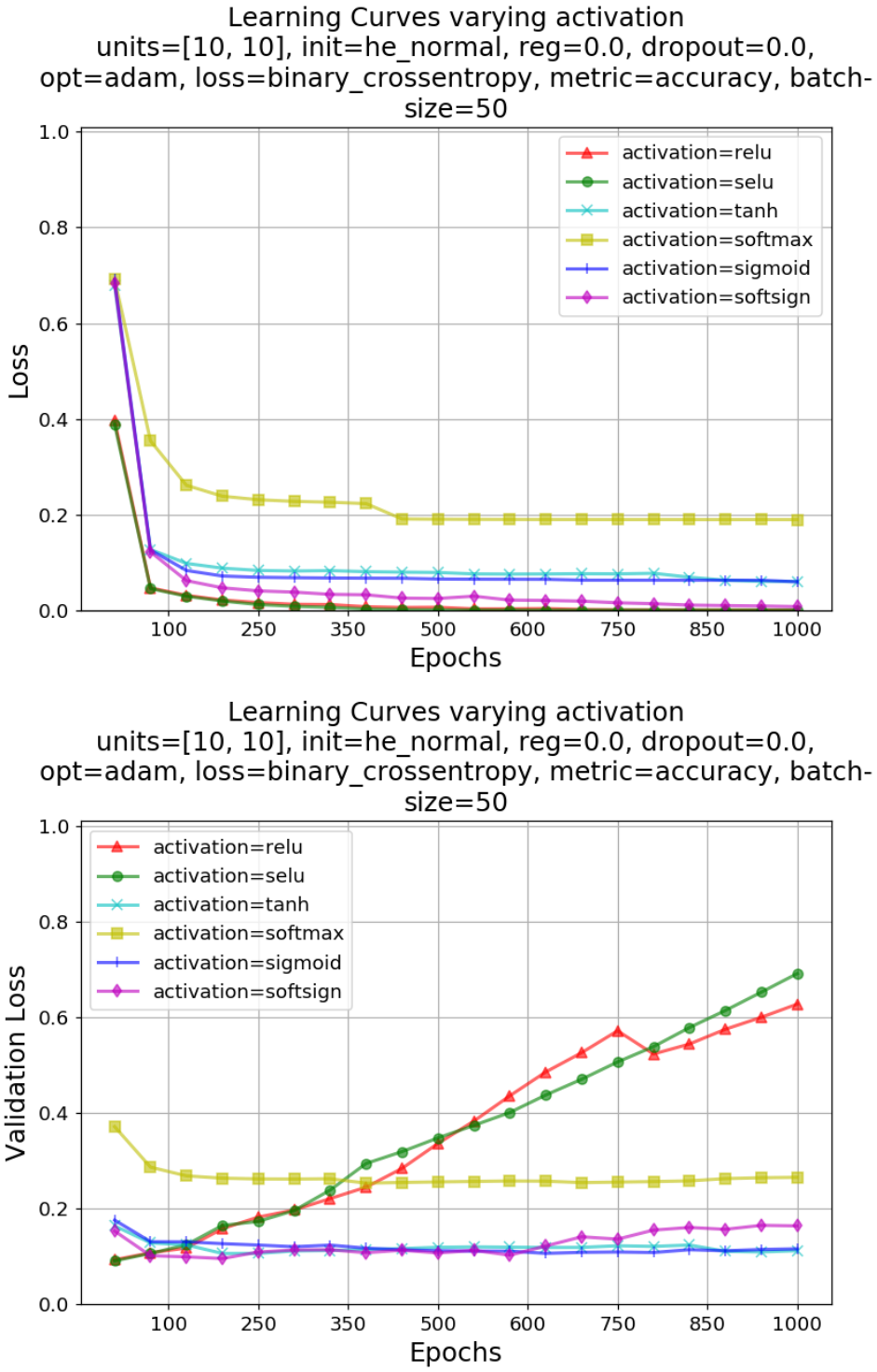


Figure 3.15: 2 hidden layers MLP benchmark for selecting the Activation Functions to use in the Bayesian Optimization grid for Synthetic Dataset 3.

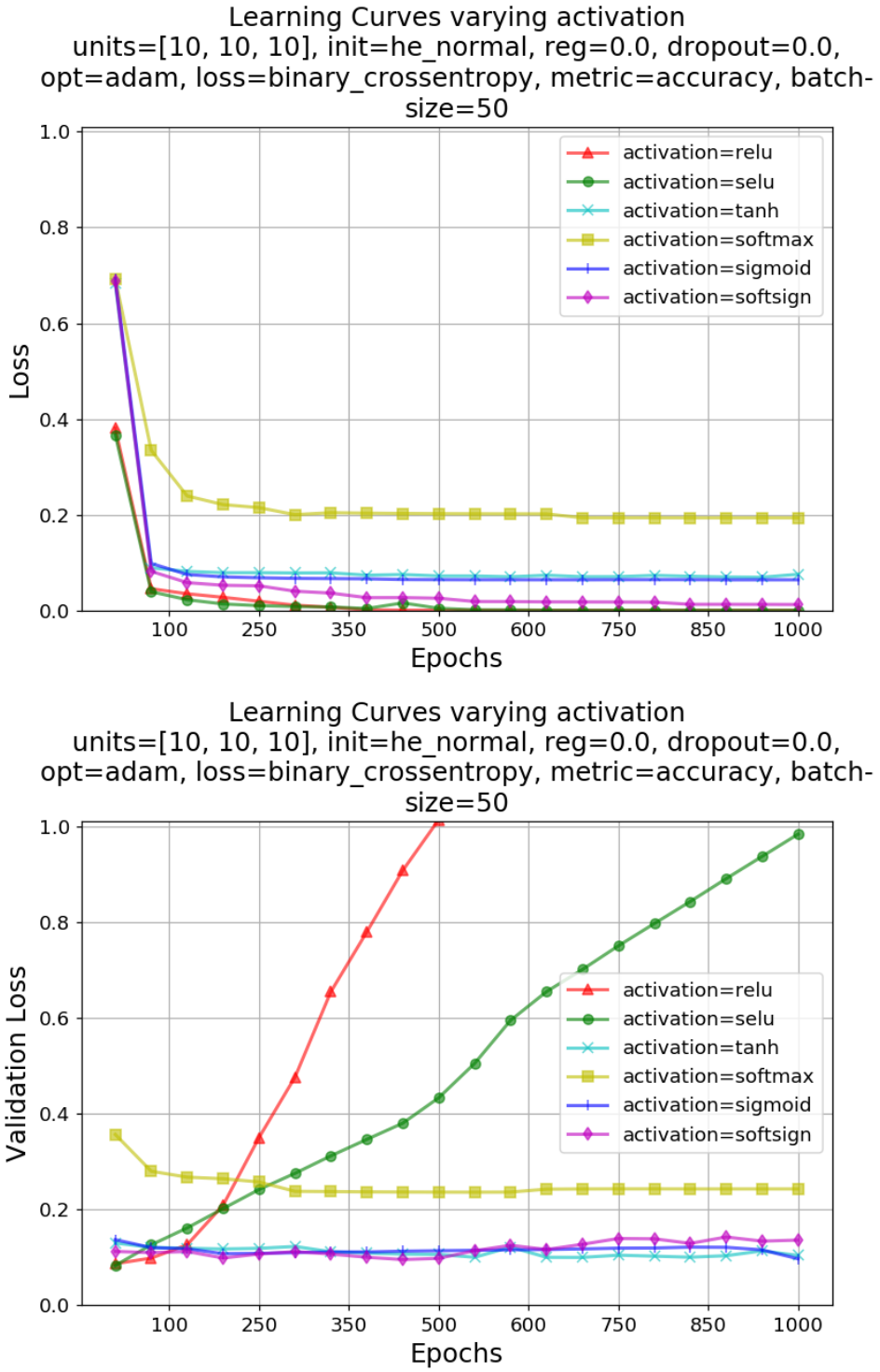


Figure 3.16: 3 hidden layers MLP benchmark for selecting the Activation Functions to use in the Bayesian Optimization grid for Synthetic Dataset 3.

The most valuable training results of the MLPs on Synthetic Dataset 3 are reported in three separate tables: Table 3.10 for 1 hidden layer, Table 3.11 for 2 hidden layers, and Table 3.12 for 3 hidden layers. The hyper-parameters inserted in these tables will be used in the next section to produce the final MLPs and to test them.

Grid Type	**Best Units per layer Hidden Layer 1**	**Hidden Layer 2**	**Hidden Layer 3**	**Best Activation**	**Best Weight Init.**	**Best Weight Reg.**	**Best Dropout Rate**	**Best 5-fold CV Acc. (%)**	**Best Val. Loss (%)**
Loose	32			Relu	Lecun Normal	0	0.4	98.889	0.051
	64			Selu	Lecun Normal	0	0.5	98.889	0.049
	64			Selu	Lecun Normal	0	0.4	98.889	0.055
	128			Relu	Lecun Normal	0	0.5	98.889	0.064
	256			Selu	Lecun Normal	0	0.4	98.889	0.055
	others							< 98.889	> 0.049
Fine	136			Relu	Lecun Normal	0	0.55	98.889	0.051
	120			Relu	Lecun Normal	0	0.5	98.889	0.053
	others							< 98.889	> 0.049

Table 3.10: The most valuable training results for Synthetic Dataset 3 (PCA retained variance = 0.95) for a MLP with 1 hidden layer.

Grid Type	Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Best 5-fold CV Acc. (%)	Best Val. Loss (%)
Loose	32	8		Selu	Lecun Normal	0	0.5	99.074	0.042
	32	8		Selu	Lecun Normal	0	0.4	98.981	0.047
	32	32		Relu	Lecun Normal	0	0.5	98.981	0.053
	32	256		Relu	Lecun Normal	0	0.5	98.981	0.046
	128	256		Relu	Lecun Normal	0	0.5	98.981	0.054
	others							< 98.981	> 0.042
Fine	others							< 98.981	> 0.042

Table 3.11: The most valuable training results for Synthetic Dataset 3 (PCA retained variance = 0.95) for a MLP with 2 hidden layers.

Grid Type	Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Best 5-fold CV Acc. (%)	Best Val. Loss (%)
Loose	64	8	8	Relu	Lecun Normal	0	0.55	99.167	0.046
	64	8	256	Relu	Lecun Normal	0	0.5	99.074	0.046
	128	8	8	Relu	Lecun Normal	0	0.4	99.074	0.049
	128	4	8	Relu	Lecun Normal	0	0.4	99.074	0.054
	64	4	4	Relu	Lecun Normal	0	0.55	99.074	0.139
	others							< 99.074	> 0.046
Fine	others							< 99.074	> 0.046

Table 3.12: The most valuable training results for Synthetic Dataset 3 (PCA retained variance = 0.95) for a MLP with 3 hidden layers.

3.4 Testing, performance evaluations and best synthetic models

In this paragraph all the five best found SVM classifiers are tested on their reduced Synthetic Test Set, reduced with the proper level of PCA. Instead, the three MLPs are tested on the reduced Synthetic Test Set of Synthetic Dataset 3.

3.4.1 SVMs testing, performance and best synthetic model

Starting with the SVM, the results of the tests are reported in Table 3.13. *Test Acc.* is the accuracy calculated on the test set, *# of Errors* are the number of errors, i.e. the sum of the mispredicticted test samples, *Error Rate (%)* is the ratio between the number of errors and the size of the test

set, which is 360, in percentage. As expected from the 10-fold CV accuracy of the training stage, the classifier trained on Synthetic Dataset 3 outperforms the others with only 1 sample mispredicted over 360, which corresponds to an error rate of 0.278%.

Synt. Dataset	**Retained Variance**	**Best (C, γ)**	**Best CV Training Acc. (%)**	**Test Acc.**	**# of Errors**	**Error Rate (%)**
1	0.90	(2.2E-02, 2.0E-04)	98.958	99.167	3	0.833
2	0.925	(7.7E-03, 1.5E-04)	99.097	99.444	2	0.56
3	**0.95**	**(7.7E-03, 1.5E-04)**	**99.167**	**99.722**	**1**	**0.278**
4	0.975	(7.2E-03, 8.7E-05)	99.028	99.167	3	0.83
5	0.99	(7.2E-03, 8.7E-05)	99.028	98.889	4	1.11

Table 3.13: Summary of the test results of the SVMs of Table 3.8. The best SVM is in bold.

The performance of the best found SVM, the one with (C, γ) = (7.7E-03, 1.5E-04), are also illustrated with some graphs.

From the Confusion Matrix in Figure 3.17 the same information of Table 3.13 is represented. In particular, it is possible to see that precision is at 1, because no False Positives are predicted, and recall is at 0.994 because of one False Negative misprediction.

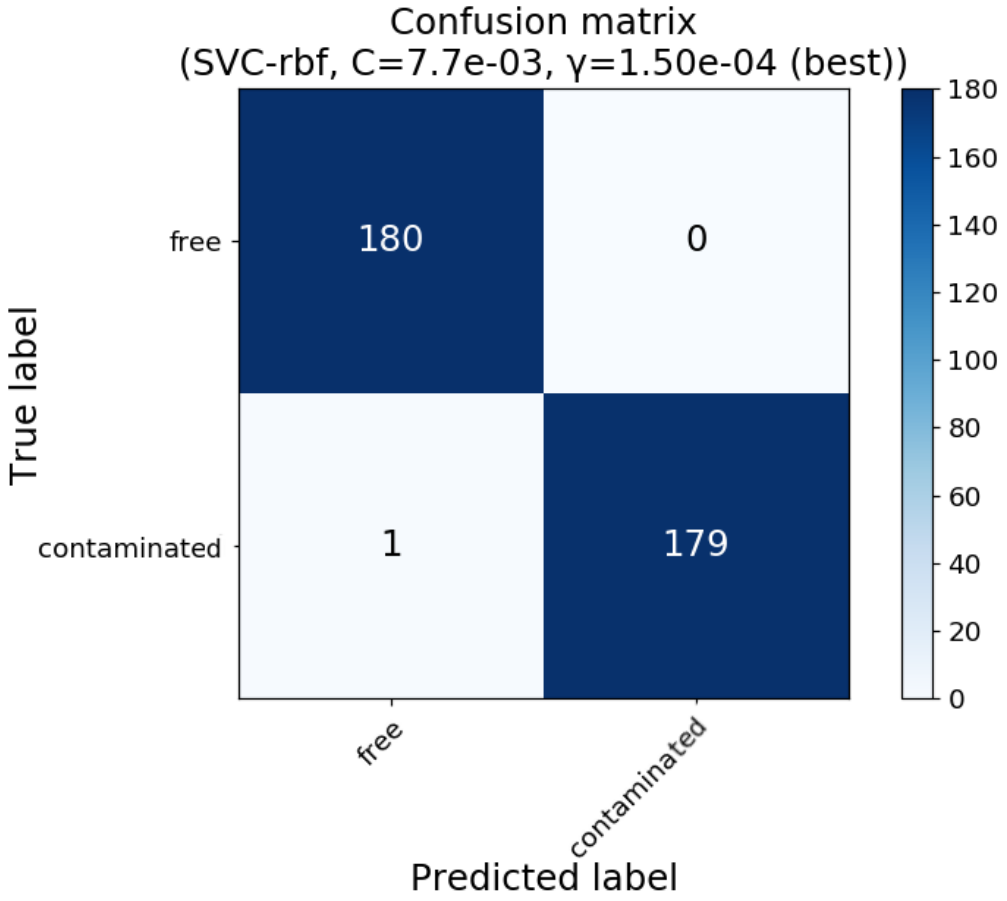


Figure 3.17: Confusion Matrix of the best found SVM, the one with (C, γ) = (7.7E-03, 1.5E-04), calculated with the corresponding Synthetic Test Set 3.

In Figure 3.18 precision and recall scores are plotted versus the decision threshold of the SVM. This graph shows how precision and recall could change with the choice of the decision threshold of the classifier. In this case, no threshold adjustment is performed [43]. Hence, the decision threshold is left at 0, which gives the same values of precision and recall cited before.

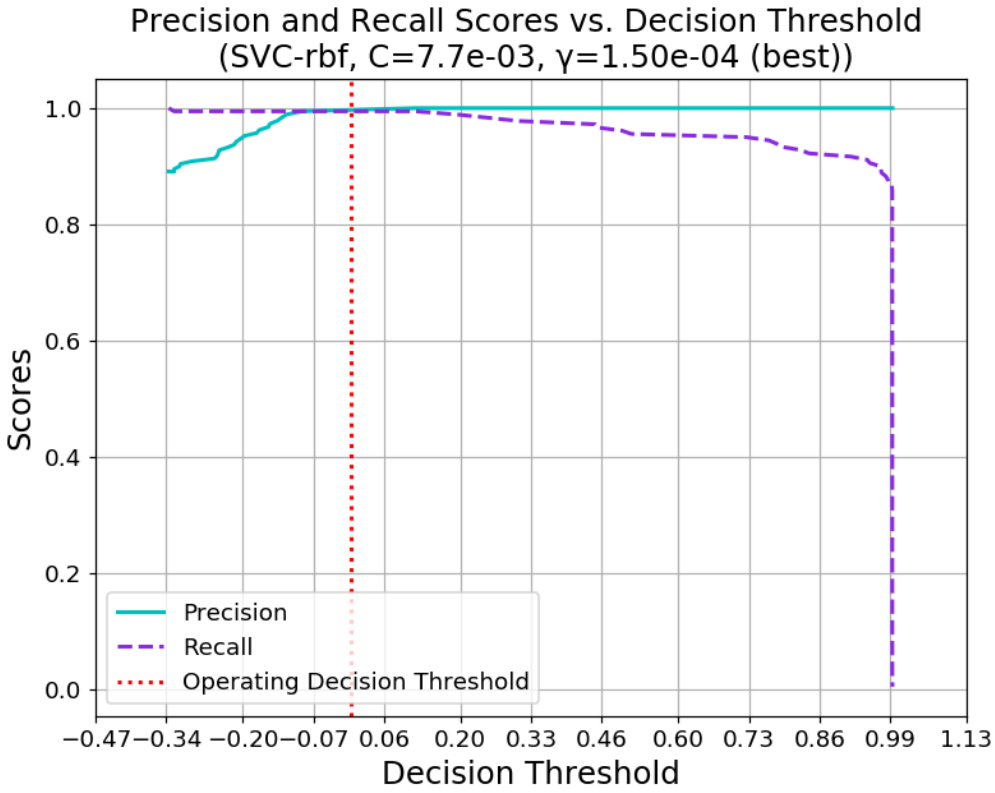


Figure 3.18: Precision and recall scores vs. decision threshold of the best found SVM, the one with (C, γ) = (7.7E-03, 1.5E-04), calculated with the corresponding Synthetic Test Set 3.

Finally, the ROC Curve is given in Figure 3.19. The operating point of this classifier is very near the perfection point placed at coordinates (0, 1). Moreover, the Area Under the Curve (AUC) is 0.999. These two features let state this is a very good classifier.

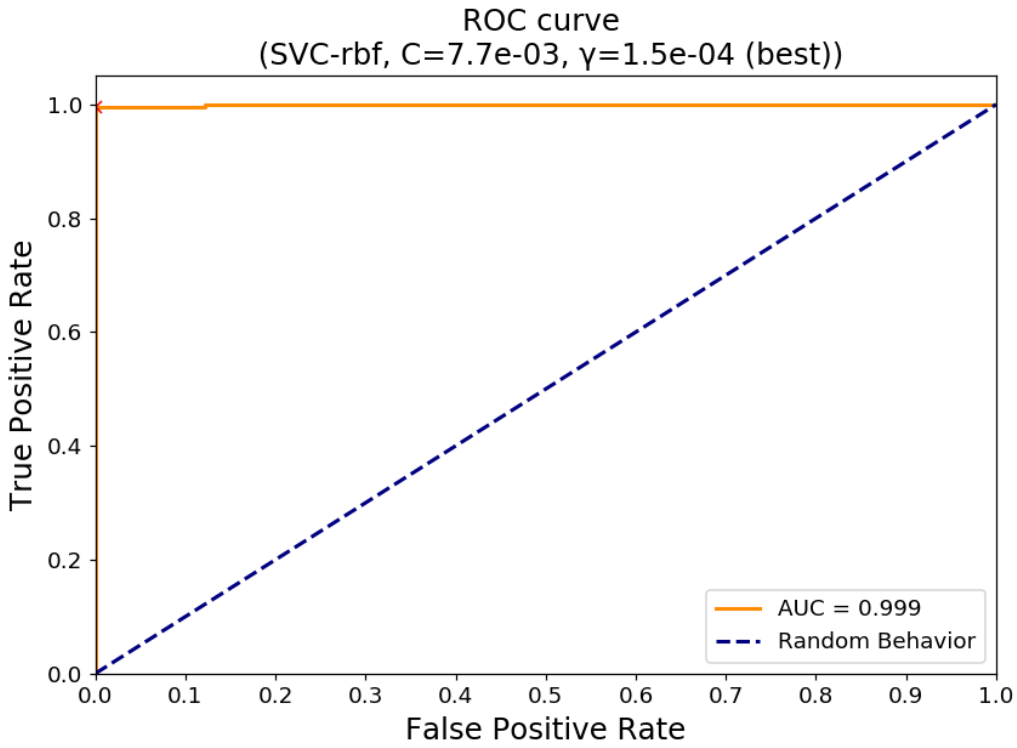


Figure 3.19: ROC Curve of the best found SVM, the one with (C, γ) = (7.7E-03, 1.5E-04), calculated with the corresponding Synthetic Test Set 3.

For this SVM classifier, 30-fold nested cross-validation (30-fold in both the inner and outer loops) is carried out on the entire Synthetic Dataset 3 (not split in 80%-20%) because this should be the common practice to present ML results (Paragraph 2.2.4). Table 3.14 shows that among the 30 folds only two couples (C, γ) are selected by Grid-Search: the first with (C, γ) = (6.0E-02, 2.6E-04) reaches (99.546 ± 0.329)% with a 95% confidence level; the second with (C, γ) = (7.7E-03, 1.5E-04), the same hyper-parameters of the 10-fold non-nested CV case, arrives to (98.542 ± 0.835)% with a 95% confidence level. The grids used for training these classifiers are the same of the 10-fold non-nested CV of Table 3.5. One could expect the results found with the 10-fold non-nested CV case should have recurred again. Instead, the solution with (C, γ) = (6.0E-02, 2.6E-04) proved that is not. The reason why this happened is because the nested approach tested the classifier with (C, γ) = (6.0E-02, 2.6E-04) on all the outer folds in which the model resulted to be the best in the relative inner folds, as discussed in 2.2.4. So the good performance of the SVM with (C, γ) = (7.7E-03, 1.5E-04) in the 10-fold

non-nested CV case could be caused by the particular choice of the test set. However, the nested and non-nested methods are both valid and depends on the will of the engineer trusting one rather than the other. A possible overcome to this stalemate is to implement both solutions and measure their performance on a new unseen dataset.

Synthetic Dataset	**Retained Variance**	**# Occurrences**	**Best (C, γ)**	**Best CV Training Acc. (%)**	**95% Confindence Interval**	**Worst Case**
3	0.95	22	**(6.0E-02, 2.6E-04)**	**99.546**	**0.329**	**99.216**
3	0.95	8	(7.7E-03, 1.5E-04)	98.542	0.835	97.706

Table 3.14: Nested-CV results of the classifier trained on Dataset 3 (PCA retained variance = 0.95). The best nested SVM is in bold.

3.4.2 MLPs testing, performance and best synthetic model

Before to enter in the real test section, it is important to explain the procedure adopted to prepare the most promising MLPs of Paragraph 3.3.1 (candidates) to the test phase. In short, the hyper-parameters found in Paragraph 3.3.2 with 5-fold CV for the three kinds of MLPs (with 1, 2 and 3 hidden layers) were used just to select the most promising candidates. Now, the same hyper-parameters are used to train new MLPs from scratch, with a different training set but with the same hyper-parameters of the candidates, until the epoch that gives the lowest validation loss is reached.

The new training set corresponds to the 75% of the original Synthetic Training Set 3, composed by 1080 samples. The remaining 25% has validation purposes. The value 25% is chosen arbitrarily because it leads to a validation set of 360 samples, equal in size to Synthetic Test Set 3. The only difference in the hyper-parameters respect the 5-fold CV training is the Batch-size that passes from 50 to 10.

The validation set is required to monitor the learning progress. In fact, the model has to be trained avoiding overfitting and when the validation loss arrives to be the minimum, the training of the MLP is stopped and the model is saved to the disk at that epoch for the subsequent testing stage. For these

reasons, the training is performed manually[4].

The information that are saved to the disk for each model are the architecture of the network, such as how many input, hidden and output layers, how many neurons per layer and which activation function they perform, and the weights of the network, which are the outcomes of the training process. So these information are a checkpoint of the model: at a later time the network can be trained more or can be used to predict new input samples. The practical adoption of these information for this thesis is for generating a C/C++ synthesizable code from them automatically, in the event that the MLP outperforms the SVM. More details in Chapter 5.

The reason why the saving procedure is not done directly during the training phase with 5-fold CV is because loading and saving a model would require too much time, just after a few number of Bayesian Optimization iterations (tens of minutes just after 30 iterations). Therefore, the training phase with 5-fold CV is used to select the most promising candidates without saving the models. In this way 150 Bayesian Optimization iterations can be reached in about one day on a normal PC.

Now the performance on the manually trained MLPs are reported in Table 3.15 for 1 hidden layer, Table 3.16 for 2 hidden layers, and Table 3.17 for 3 hidden layers. Every classifier is tested on Synthetic Test Set 3. A new column, *Epochs*, is added compared to Table 3.13: it indicates the epochs at which the manual training was stopped.

The most performing MLPs are summarized in Table 3.18. The winning classifier is highlighted in bold. It is a 3 hidden layers MLP, which mispredicts 4 out of 360 samples, so it has an error rate of 1.111%. Therefore, the clues got after the 5-fold CV training phase are confirmed: the best MLP found so far needs 3 hidden layers.

[4]Although this training is called "manual", a model is programmed to be saved automatically every time its validation accuracy overcomes the maximum validation accuracy found until that epoch. The human intervention is only for stopping the training when the model start to overfit or when it doesn't learn anymore.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Epochs	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	# of Errors	Error Rate (%)
32			**Relu**	Lecun Normal	0	0.4	66	**98.889**	**0.055**	**98.333**	**6**	**1.67**
64			Selu	Lecun Normal	0	0.5	14	98.889	0.060	97.778	8	2.22
64			Selu	Lecun Normal	0	0.4	22	98.889	0.061	97.222	10	2.78
128			Relu	Lecun Normal	0	0.5	14	98.889	0.056	98.056	7	1.94
256			Selu	Lecun Normal	0	0.4	22	99.167	0.060	97.222	10	2.78
136			Relu	Lecun Normal	0	0.55	22	99.167	0.052	98.056	7	1.94
120			Relu	Lecun Normal	0	0.5	6	98.889	0.055	98.056	7	1.94

Table 3.15: Test results on Synthetic Test Set 3 of the manually trained MLPs with 1 hidden layer. The best MLP is in bold.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Epochs	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	# of Errors	Error Rate (%)
32	**8**		**Selu**	Lecun Normal	0	0.5	104	**99.167**	**0.046**	**98.056**	**7**	**1.94**
32	**8**		**Selu**	Lecun Normal	0	0.4	74	**99.167**	**0.045**	**98.056**	**7**	**1.94**
32	32		Relu	Lecun Normal	0	0.5	186	99.160	0.046	97.778	8	2.22
32	256		Relu	Lecun Normal	0	0.5	88	98.889	0.060	97.778	8	2.22
128	**256**		**Relu**	Lecun Normal	0	0.5	86	**98.889**	**0.059**	**98.056**	**7**	**1.94**

Table 3.16: Test results on Synthetic Test Set 3 of the manually trained MLPs with 2 hidden layers. The best MLPs are in bold.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Epochs	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	# of Errors	Error Rate (%)
64	8	8	Relu	Lecun Normal	0	0.55	236	99.167	0.051	98.333	6	1.67
64	8	256	Relu	Lecun Normal	0	0.5	158	99.167	0.046	96.944	11	3.06
128	8	8	Relu	Lecun Normal	0	0.4	34	99.167	0.047	98.333	6	1.67
128	**4**	**8**	**Relu**	Lecun Normal	0	0.4	54	**99.167**	**0.050**	**98.889**	**4**	**1.111**
64	4	4	Relu	Lecun Normal	0	0.55	360	98.889	0.143	98.333	6	1.67

Table 3.17: Test results on Synthetic Test Set 3 of the manually trained MLPs with 3 hidden layers. The best MLP is in bold.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	# of Errors	Error Rate (%)
32			Relu	98.889	0.055	98.333	6	1.67
32	8		Selu	99.167	0.046	98.056	7	1.94
32	8		Selu	99.167	0.045	98.056	7	1.94
128	256		Relu	98.889	0.059	98.056	7	1.94
128	**4**	**8**	**Relu**	**99.167**	**0.050**	**98.889**	**4**	**1.111**

Table 3.18: Summary of the test results of the MLPs in Table 3.15, 3.16 and 3.17. The best MLP is in bold.

The performance of the MLP with 3 hidden layers and [128, 4, 8] Relu Units, the one which gave the lowest error rate and which is reported in bold in Table 3.18, are illustrated with some graphs.

Before that, the Loss Curves and the Learning Curves of this classifier are plotted in Figure 3.20. The Training Loss decreases fast for the first 100 epochs, then continues to diminish very little and stays below 0.1: this is an index the model is learning from the training data. For what concerns the Validation Loss, it follows the Training Loss for 300 epochs, then it starts to increase, showing the first symptoms of overfitting. This is not a problem because, as reported in Table 3.12, this classifier is stopped and saved at epoch 54 which corresponds to one of the minima of the Validation Loss. Not by chance the peak of the Validation Accuracy (99.167%) occurs at the same epoch.

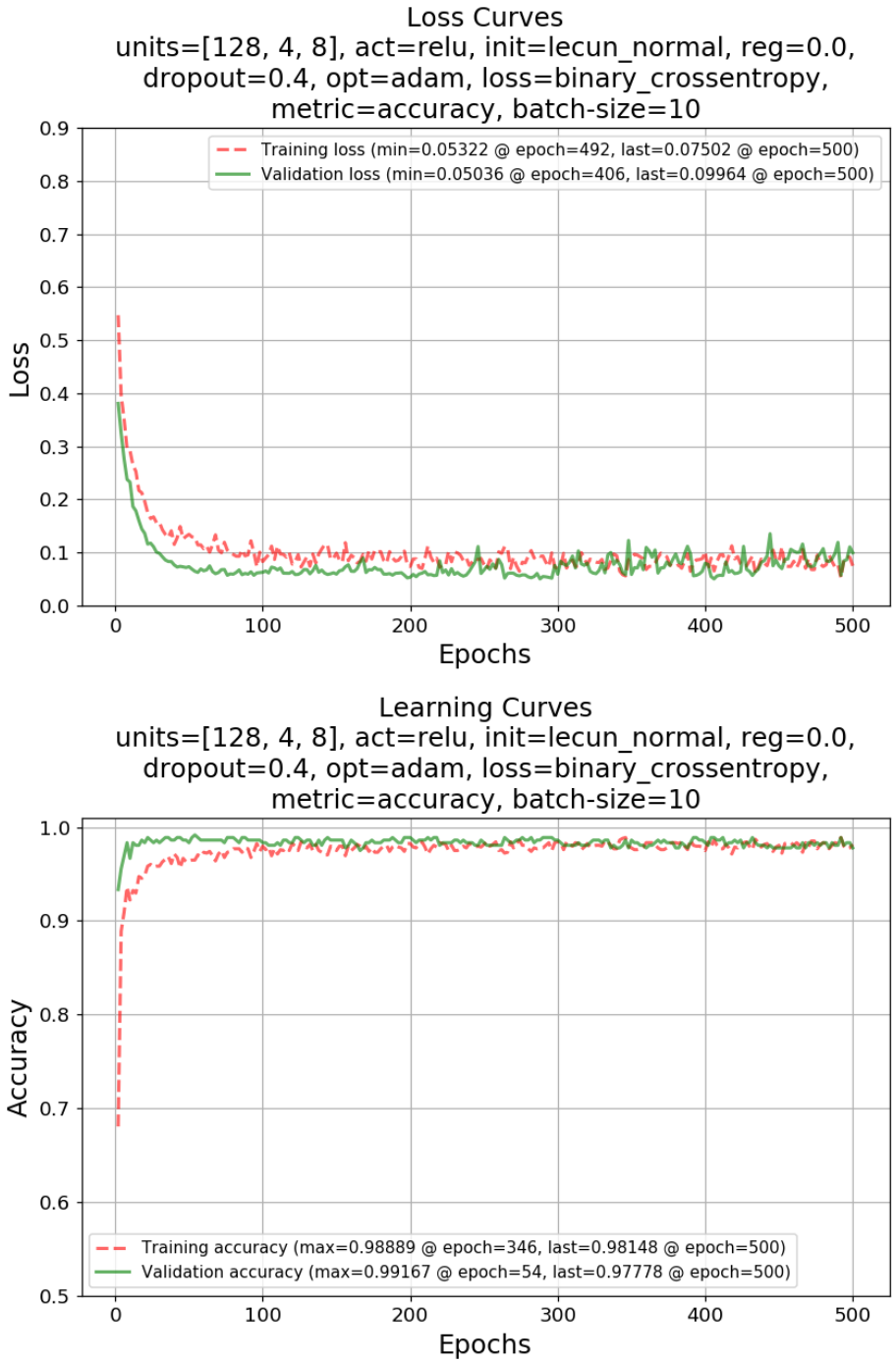


Figure 3.20: Loss Curves and the Learning Curves of the best found MLP with 3 hidden layers and [128, 4, 8] Relu Units, trained with 75% of the training set of Synthetic Dataset 3 and validated on the remaining 25%.

Now, the performance metrics. The Confusion Matrix can be seen in Figure 3.21. Precision is at 1, because no False Positives are predicted, and recall is at 0.978 because of four False Negative mispredictions.

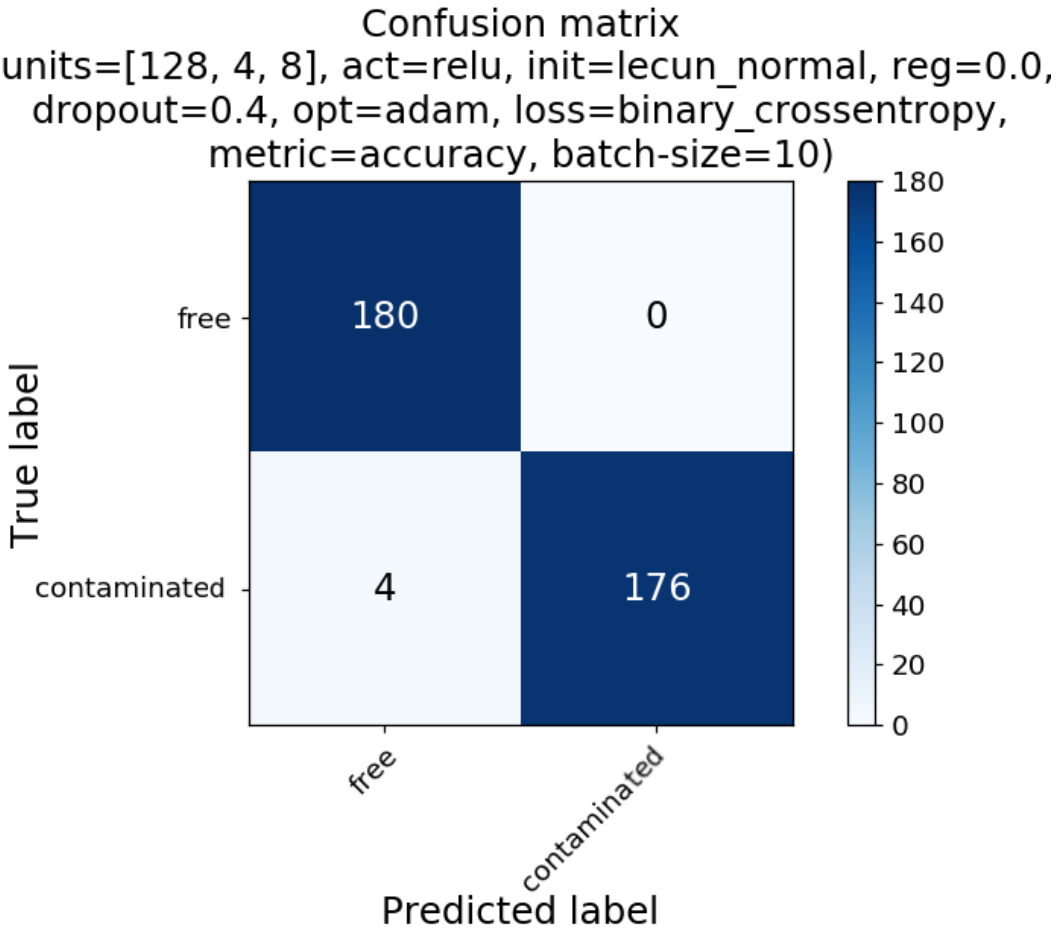


Figure 3.21: Confusion Matrix of the best found MLP with 3 hidden layers and [128, 4, 8] Relu Units, calculated with Synthetic Test Set 3.

In Figure 3.22 Precision and recall scores are plotted versus the decision threshold of the MLP. Since the output layer of this classifier is a sigmoid, it is decision threshold is 0.5 by default, as denoted by the red dotted vertical line, and the possible values span the [0, 1] range. Thus, when the output of the sigmoid activation is below the threshold, the input sample is classified as free, otherwise as contaminated. For this classifier threshold adjustment could improve the recall while maintaining maximum precision.

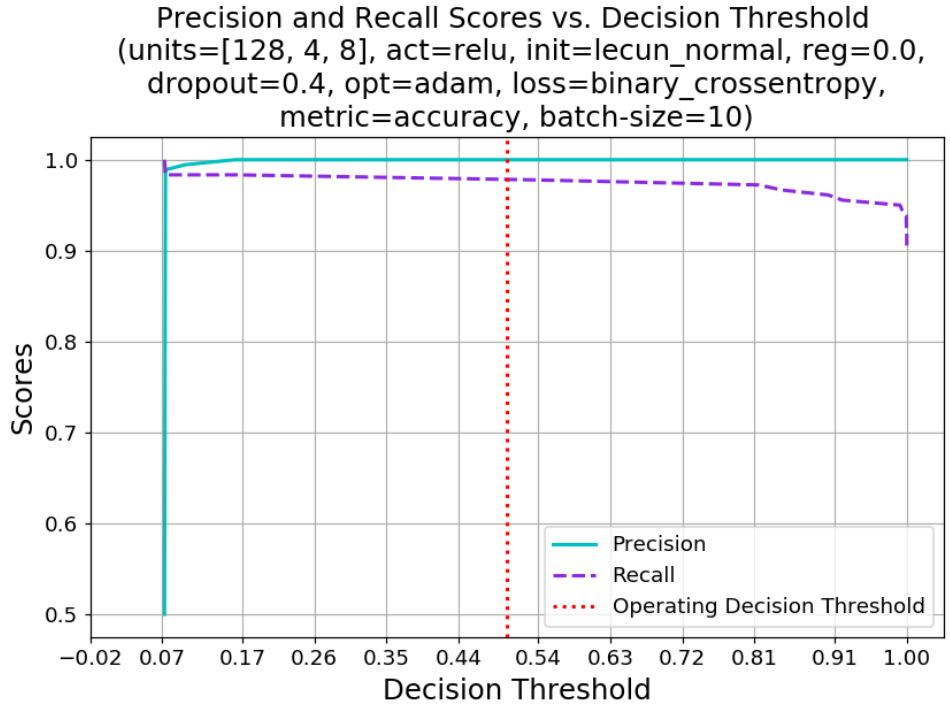


Figure 3.22: Precision and recall scores vs. decision threshold of the best found MLP with 3 hidden layers and [128, 4, 8] Relu Units, calculated with Synthetic Test Set 3.

To conclude, in Figure 3.23 the ROC Curve is plotted. The operating point of this classifier is again near the point (0, 1) and this time the AUC is 0.992, a bit less than the AUC of the best SVM found earlier. However, this classifier is still good.

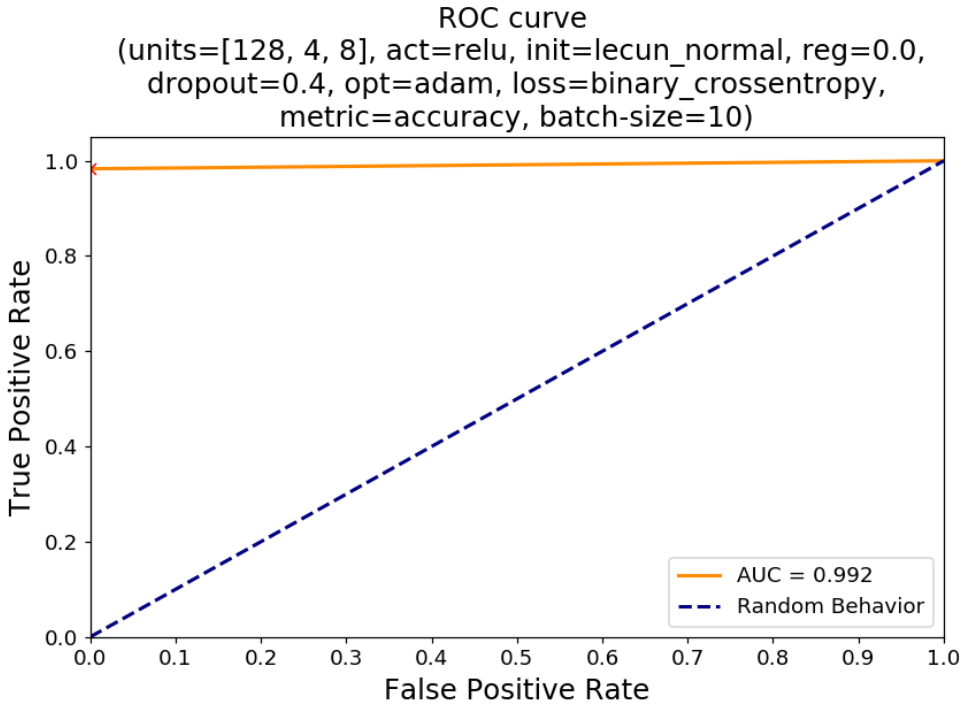


Figure 3.23: ROC Curve of the best found MLP with 3 hidden layers and [128, 4, 8] Relu Units, calculated with Synthetic Test Set 3.

For the MLP nested cross-validation accuracy is not performed at all because it is too time consuming. Indeed, MLPs are usually trained with non-nested CV. Another option is to use only a validation set, but to compensate the absence of cross-validation the training is repeated for a certain number of times with different random weight initialization due to the stochastic nature of neural networks: in fact the model can hit potential local minima depending on the initial value of its weights. On the other hand, cross-validation offers a sort of robustness the more folds are employed, as already described in 2.2.4. Ultimately, as a rule of thumb, as long as the training results are confirmed on the test set, that is there is no big difference in the classification accuracies, the training can be considered well done.

Chapter 4

Training and testing with Real Data

Due to the fact that the results of the application of ML algorithms to the Synthetic Dataset were promising, this chapter deals with the generation of a Real Dataset (the second cited in Paragraph 1.3), and the following training and testing phases of SVM and MLP classifiers.

At the beginning, some of the laboratory instruments specifications are illustrated and the real antennas arch is shown. This time the Real Dataset is created with the scattering parameters of the network composed by a safflower oil jar (which emulates the hazelnut-cocoa spread), the antennas arch and the air in between, acquired with the *MIT-Food prototype* described in Paragraph 1.3.

For the preprocessing stage, solely standardization is applied, while PCA is not needed because proved to lower the classification accuracy the less information is retained. Therefore, the standardized but non-reduced dataset is the only one employed in the next phases.

All the subsequent activities consist in training and testing SVM and MLP classifiers. They are carried out with the same procedures of the Synthetic Dataset detailed in Paragraph 3.3.

To conclude, the classification performance of the best real SVM and MLP models are evaluated on the test set and the results are compared thanks to the usual metrics.

4.1 Real Dataset creation

4.1.1 MIT-Food system prototype

As introduced in Paragraph 1.3, the Real Dataset is acquired with a series of laboratory measurements made with the *MIT-Food prototype*, illustrated in Figure 1.2. Even if this kind of system can be used to generate tomographic images, it is employed wtih ML to predict the membership of a sample in a class without create an image. For this reason it should be called *Microwave Sensing system* instead of *Microwave Imaging system*.

The general view of the *MIT-Food prototype system* is already summarized in Paragraph 1.3. So, in this section the specifications of some of its blocks are formalized.

- **Antennas arch**. The simulated model of Figure 3.1 has become reality and a picture of it can be seen in Figure 4.1. Its geometrical characteristics are the same of those declared in Paragraph 3.1.1. The support of the antennas is realized with resin and its design is suitable for an in-line assembly, on top of a conveyor belt.

  About a real triangular aperture PCB antenna:

  – its behavior can be affected on the hand-made welding between the antenna and a coaxial cable connected to its port, as shown by the different reflection coefficients in Figure 4.2;

  – the simulated antenna radiate at about 9.6 GHz as demonstrated in Figure 3.5, while the real antenna in Figure 4.2 has radiation peaks in the range $[10 \div 10.6]$ GHz. In addition, the measured reflection coefficient is better than the simulated of at least 10 dB;

  – the MWI problem is not antenna dependent as long as the simulated scenario is carefully designed to work with the frequency that is irradiated by the antenna, in this case 10 GHz. This will be even more true when a new and more accurate simulated scenario will be created which will model the jar and the antennas (currently the antennas are not simulated) to create a brand new operator $\mathcal{L}$. Further, the MWI approach is differential, so the resulting Scattering-matrix can be considered as due to the intrusion only, plus the obvious environmental noise; consequently the only limitation is the strength of the signal due to the intrusion which has to be higher than the noise level.

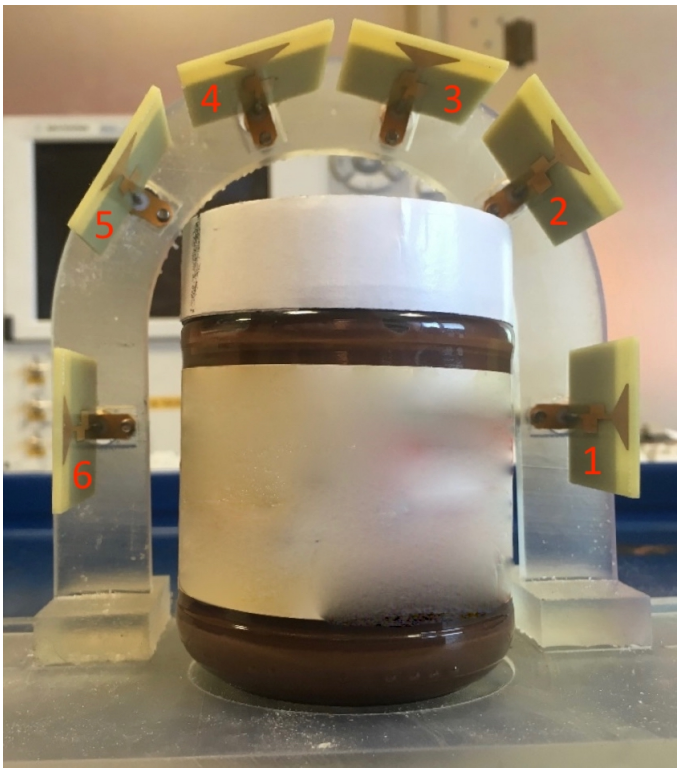


Figure 4.1: Real antennas arch with hazelnut-cocoa cream underneath. Courtesy of Ricci M. [44].

- **Vector Network Analyzer (VNA)**. It works at same frequency of the simulated case, that is 10 GHz because of the advantageous trade-off among penetration depth, resolution and electronics cost examined in Paragraph 3.1.1. Its precise name is *P9375A Keysight Streamline USB Vector Network Analyzer*[1], 26.5 GHz. Its key features are:
  - compactness, portability, and easy connections;
  - wide frequency range from 300 kHz up to 26.5 GHz;
  - support of Electronic Calibration Modules for simple and quick calibration;
  - dynamic range up to 115 dB[2];

[1] `https://www.keysight.com/en/pdx-2916424-pn-P9375A/keysight-streamline-series-usb-vector-network-analyzer?cc=US&lc=eng`

[2] Dynamic range = source maximum output power minus receiver noise floor @ 10 Hz IF bandwidth. Does not include single module crosstalk effects

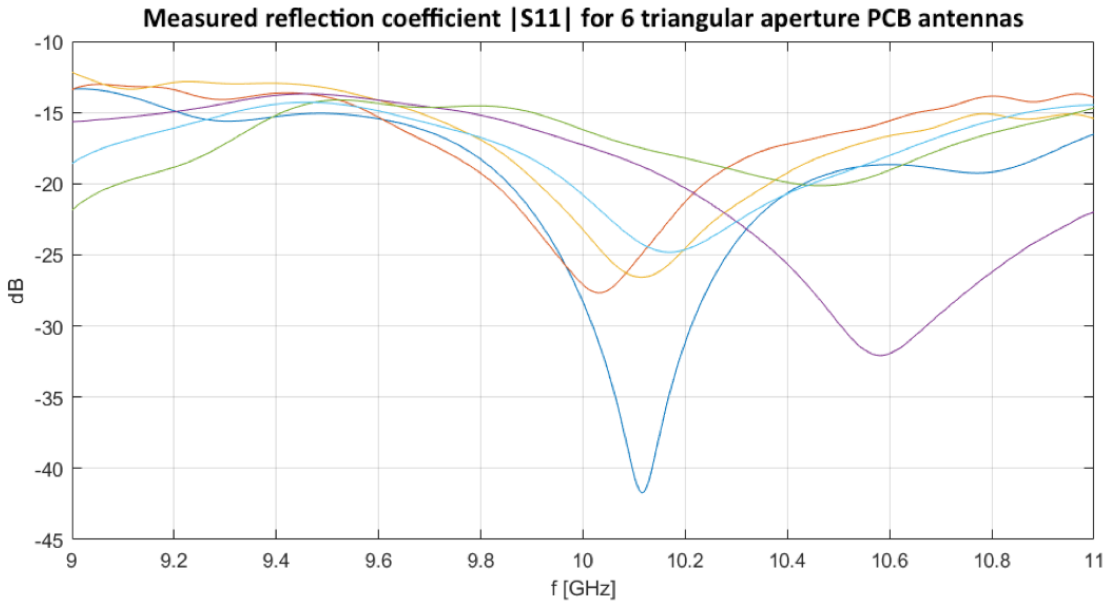


Figure 4.2: Measured reflection coefficients of the 6 triangular aperture PCB antennas. The discrepancies are caused by the different hand-made welds between the antennas and the coaxial cables.

  - measurement speed in the order of few milliseconds, with 24 ms for 201 points, full 2-port cal, and 100 kHz IF bandwidth.

- **Switching-matrix**. Despite the simplicity of its task, it is a complex subsystem. In order to be able to measure all the 30 pairs by all the combinations of the six antennas[3], a switching system piloting the antennas transmissions and receptions is necessary. The 2x6 electro-mechanicaI switching matrix is built by merging six 2-to-1 *Keysight 8762B Coaxial Switches* and two 6-to-1 *Keysight 87206B Multiport Coaxial Switches*, together with the *Driver Board L4445A* and the custom *Extender and Distribution Boards*, as shown in the block diagram of Figure 4.3. A *Matlab* script controls both the VNA and the switching matrix *Driver Board* in order to measured the 30 scattering parameters, by activating any transmitting path from one VNA port to one transmitting antenna, and any receiving path from one receiving antenna to the other VNA port.

[3]The self-scattering parameters on the main diagonal are not acquired.

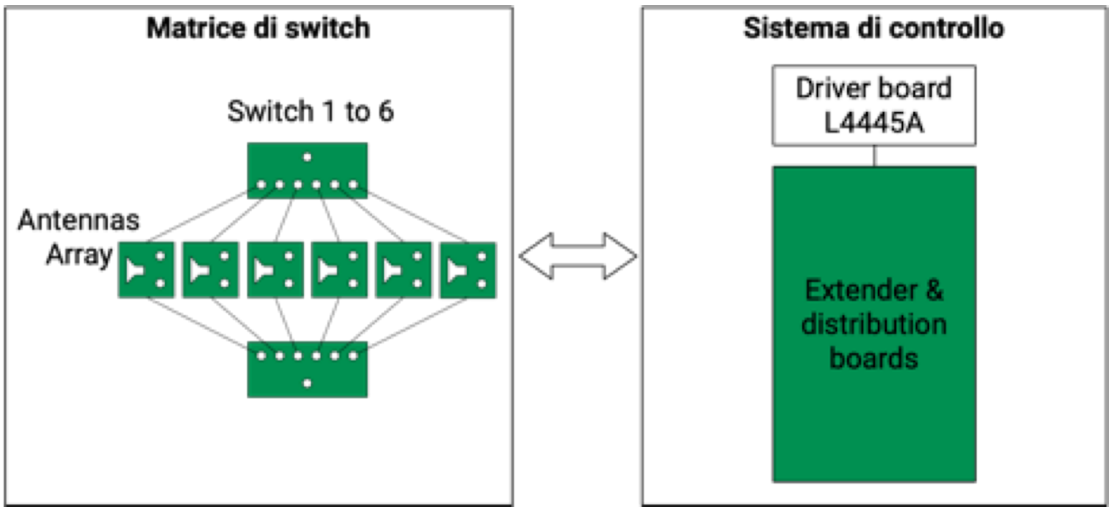


Figure 4.3: Top view of the switching matrix subsystem. Courtesy of Turvani G.

The main characteristics of these electromechanical switches are:

- small package size and portability;
- insertion loss less than 0.5 dB @ 10GHz;
- isolation between ports greater than 90 dB @ 10GHz;
- standing wave ratio (SWR) less than 1.35 dB[4];
- minimum number of cycles of 1 000 000 for model *8762B* and 5 000 000 for model *87206B*;
- maximum switching speed of 30 ms for model *8762B* and 15 ms for model *87206B*.

4.1.2 Real Dataset creation procedure

In Paragraph 1.3 it was explained the purpose of the controller: thanks to a laptop with a *Matlab* script, it orders the measurement system to acquire the Scattering-matrix of the network composed by the jar under the antennas arch, the antennas arch and the air in between, and saves them in a textual format. Instead of calculating the difference between the measured

[4]SWR is defined as the ratio of the maximum amplitude of a standing wave in a transmission line to the minimum amplitude the AC signal along that line.

Scattering-matrix and that of the reference scenario, i.e. "golden" matrix in which no intrusion is in the jar, and then solving the linear (ill-posed) inverse problem with Equation 2.4, it is decided to feed directly the ML classifiers with the acquired scattering parameters. If ML algorithms will return good results, the latency of the *MIT-Food prototype* could be reduced substantially because the prediction will be shown in real-time without the need to reconstruct a tomographic image: the prototype will become a *Microwave Sensing system*. In a final industrial application the laptop will be replaced by a compact embedded device.

Paragraph 1.3 introduced that the Real Dataset is formed by 2400 samples of scattering parameters of real measurements: 1200 contaminated and 1200 uncontaminated safflower oil jars[5]. Also this time the dataset is balanced. It is important to underline that these measurements are made in static conditions, that is the jar is not moving while the acquisition is in progress. The analysis in motion is foreseen to be performed in the next few months.

Now the Real Dataset generation procedure is described. Different types of objects are selected to be intrusions: a metal sphere, a glass fragment, a big plastic sphere, a small plastic sphere, a triangular plastic fragment, and a cap shape plastic. All of them are collected in Figure 4.5 and reported in Table 4.1 with their dimensions. For a set of contaminated samples of the same type, the steps are:

1. **prepare the selected contaminant**: the target is closed in a latex glove or in a plastic net knotted with a fishing wire to manage it easier from outside the jar;

2. **insert the contaminant in the jar**: when the position of the intrusion inside the jar is satisfactory, the fishing wire is stuck externally to the jar with adhesive tape to keep the target in position. For the first 100 samples the target is put more or less at the half height of the jar. For the second 100 samples it is left floating in surface. In the case the specific weight of the contaminant does not allow it to sink into the oil, or in other words it can only stays floating in surface, 200 samples per

[5]The safflower oil replaces hazelnut-cocoa cream because it is transparent, and so the position of the contaminants can be monitored easily, and its behavior in the microwave spectrum is similar to the chocolate spread, since $\varepsilon_{oil} \simeq \varepsilon_{cream} = 2.86$ @ 10 GHz. The jar is made of transparent plastic and it is the original Nutkao's vessel.

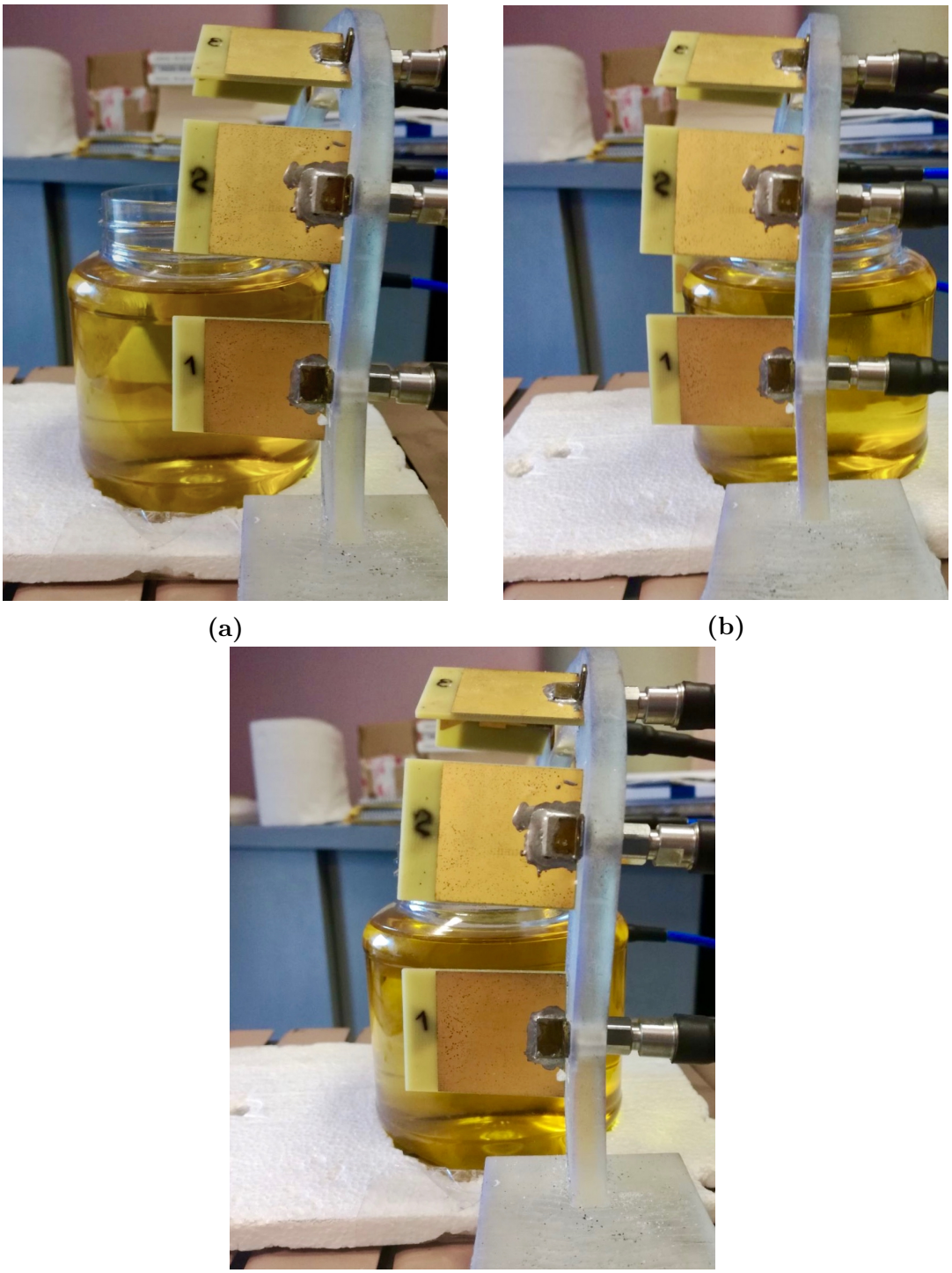
intrusion type are acquired. Instead, the position of the target in the horizontal plane (at fixed height) is changed about every 25 samples;

3. **place the jar under the antennas arch**: the jar is placed approximately inside the range $\pm$ 2 cm from the central position under the antennas (Figure 4.4) arch and is rotated of a maximum angle of $\pm 10°$ with respect to its long-side axis. The non perfect position is desired to take into account the possible non precise acquisition time instant and the not perfectly straight orientation of the jar when it will move on the conveyor belt. At every sample, the jar is moved and rotated, always remaining inside the range and the maximum angles defined above;

4. **run the acquisition from the controller**: the *Matlab* script is executed, the switching matrix and the VNA compute the measurements and the Scattering-matrix is obtained;

5. **convert the S-matrix in a sample**: since the S-matrix is symmetric due to reciprocity, only the elements placed in the triangular upper part of the matrix are used because the other would be redundant, as already mentioned in Paragraph 2.1. Moreover, as every scattering parameter is a complex numbers, row-by-row these elements are flattened in a vector of real numbers, where the real and the imaginary parts are placed one after the other. The elements on the main diagonal, the self-scattering parameters of the antennas, are not included also because they produced worse results in preliminary image reconstruction experiments. So the resulting sample has a length of 30, i.e. 30 features.

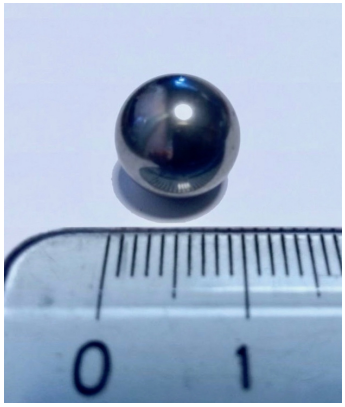
On the other hand, to make an uncontaminated/free sample, uniquely steps 3, 4 and 5 are necessary.



(a) **(b)**

(c)

Figure 4.4: The central position below the antennas arch in c) and the two extreme positions that the jar can assume during the measurements, -2 cm in a) and +2 cm in b).

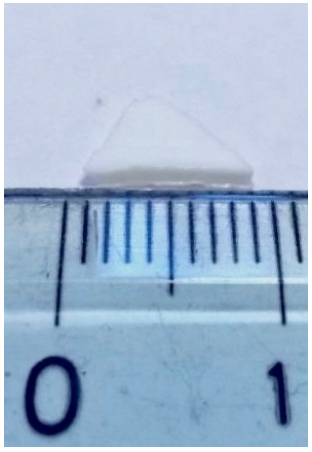


(a) Metal sphere.

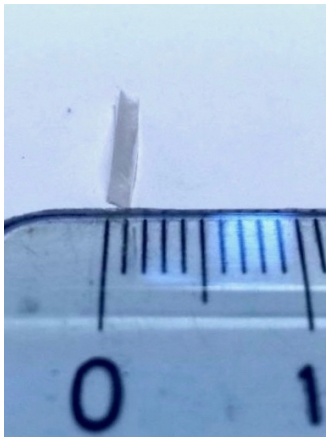
(b) Glass fragment.

(c) Big plastic sphere (in a plastic net knotted with a fishing wire).

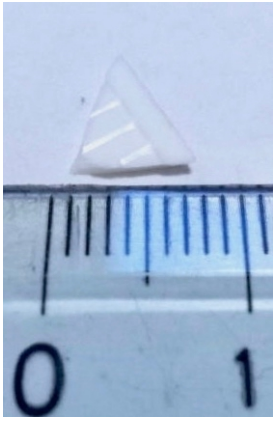
(d) Small plastic sphere (in a latex glove knotted with a fishing wire).



(e) Triangular plastic fragment (view 1).



(f) Triangular plastic fragment (view 2).



(g) Triangular plastic fragment (view 3).

(h) Cap shape plastic (view 1).

(i) Cap shape plastic (view 2).

Figure 4.5: Pictures of all types of contaminants.

In Table 4.1 all types of contaminants employed to populate the Real Dataset are listed. $Max.\ Size(mm)$ and $Min.\ Size(mm)$ are the lengths of the maximum and minimum dimensions of the target in millimeters.

Type of Contaminant	**# of Samples at Half Height**	**# of Samples in Surface**	**Total # of Samples**	**Max. Size (mm)**	**Min. Size (mm)**
Metal sphere	100	100	200	10	
Glass fragment	100	100	200	13	2
Big plastic sphere	100	100	200	20	
Small plastic sphere	100	100	200	3	
Triangular plastic fragment	200		200	8	1
Cap shape plastic	200		200	15	9

Table 4.1: All types of contaminants contained in the Real Dataset. Measurement uncertainty: 1 mm.

In the case of the Real Dataset, at the opposite of the Synthetic Dataset, the replicability of the generation process is impossible due to the intrinsic nature of a scientific experiment.

4.2 Real Dataset preprocessing

For the preprocessing stage, solely standardization is applied. About PCA, it is not used because gave lower classification accuracy the less information was retained. Therefore, this time the initial Real Dataset is standardized, but non-reduced, so there is only one dataset for the next phases, instead of five of the synthetic case.

The preprocessing stage of the Real Dataset is very similar to the one of the Synthetic Dataset of Paragraph 3.2. Indeed, the Real dataset is shuffled and then split in two parts, a training set of 80% of the data and a test set of 20%. Again, the number of samples belonging to the "contaminated" class and the "free" class are equal in both training and test sets, so they are balanced.

Then, the same scaling of the synthetic case, standardization, is applied to the training and test set. However, since the highest feature in the Real Dataset is 0.004928 and the lowest is -0.007949, scaling may not be necessary because all the features have already the same order of magnitude. This

statement is verified by training SVMs and MLPs with a non-standardized Real Dataset with the identical procedures followed in the previous section, Paragraph 3.3. Here only the final results are reported: the 10-fold CV accuracy of the best non-standardized SVM is about 1% less than the one of the best standardized SVM; the 5-fold CV accuracy of the best non-standardized MLP is around 10% lower than the respective accuracy of the best standardized MLP. Therefore, standardization definitely helps.

Talking about PCA for features reduction, this time it is not used due to the already low number of features of a real sample, which is 30. Thus, there is one Real Dataset usable for training. This choice is validated by training five different SVMs with as many Real Datasets reduced by keeping the same amounts of retained variance of the synthetic case: 0.9, 0.925, 0.95, 0.975, and 0.99. The training stopped at Loose Grid 2 because the purpose was to find a trend and not a precise result. So the best 5-fold CV accuracies after Loose Grid 2 for each Real Dataset are shown in Table 4.2. Clearly, the more reduction is performed, the more information is lost, affecting the accuracy. In fact, in the last row, the original Real Dataset (with no PCA reduction) provides the highest accuracy. The conclusion is that a feature reduction method like PCA is convenient when the number of features is extremely high, i.e. in the order of thousands, according to [29].

Synthetic Dataset	**Dataset Size (100%)**	**Retained Variance**	**# of Extracted Features**	**Synthetic Training Set Size (80%)**	**Synthetic Test Set Size (20%)**	**5-fold CV Acc. (%)**
1	2400	0.90	10	1920	480	86.406
2	2400	0.925	11	1920	480	86.667
3	2400	0.95	14	1920	480	90.313
4	2400	0.975	14	1920	480	91.354
5	2400	0.99	22	1920	480	92.344
6	2400	1.00	30	1920	480	94.531

Table 4.2: The best 5-fold CV accuracies after Loose Grid 2 for each SVM trained with a Real Dataset reduced with an increasing retained variance. The last row reports the accuracy of the SVM trained with the original Real Dataset (with no PCA reduction).

As already done in Paragraph 3.2, the 2-D and 3-D representations of the Real Dataset are illustrated in Figure 4.6a and 4.6b: in red the contaminated samples, in green the uncontaminated/free ones. These plots are obtained by

reducing that dataset with PCA with the constraint to have respectively 2 and 3 principal components as outcomes. It is important to remember these graphs are just informative.

4.3 Training procedures and real candidate models

The training procedures that are adopted for the 2 kinds of classifiers taken into account, SVMs and MLPs, are identical to those described in Paragraph 3.3 for the synthetic case. Hence, this section addresses to report and comment the training results of the classifiers on the Real Dataset.

4.3.1 SVMs training and real candidate models

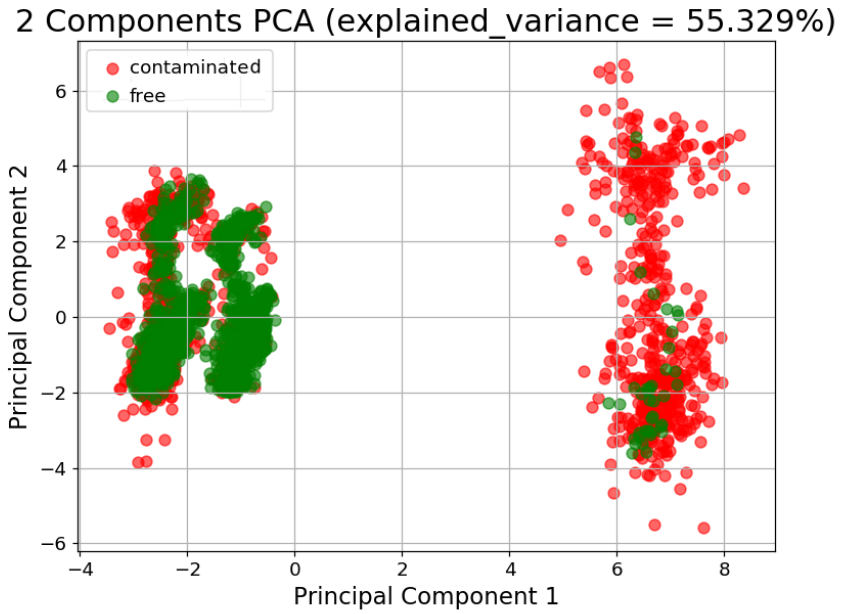
The SVMs are trained on the Real Dataset with the procedure discussed in Paragraph 3.3.1. The training results with the highest accuracy are reported in Table 4.3. The most performing SVMs are three with a 10-fold CV accuracy of 95.052%, but the one with the lower C is selected and it is in underlined in bold. This SVM will be tested in Paragraph 4.4.1.

Grid Type	**# Folds**	**C Grid**	**γ Grid**	**Best (C, γ)**	**Best CV Training Acc. (%)**
Loose 1	5	[-5, 10] 16p	[-15, 3] 19p	(1.0E+02, 1.0E-02)	93.958
				(1.0E+04, 1.0E-03)	93.958
Loose 2	5	[0, 8] 18p	[-6, 0] 14p	(2.3E+02, 1.4E-02)	94.531
				(7.6E+01, 4.1E-02)	94.531
				(2.6E+01, 4.1E-02)	94.531
Fine	**10**	[1, 3] 9p	[-3, -1] 15p	**(5.6E+01, 2.7E-02)**	**95.052**
				(1.8E+02, 1.4E-02)	95.052
				(1.0E+02 , 1.9E-02)	95.052

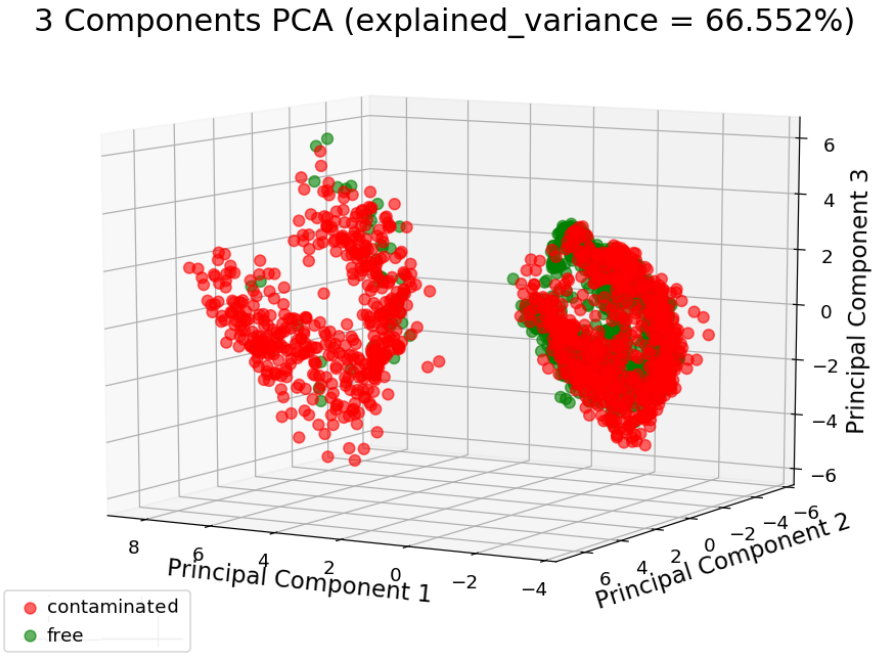
Table 4.3: SVM training results for the Real Dataset. The best (C, γ) and its CV accuracy are in bold.

In addition, the plots that are produced by the training procedure can be seen in the following figures.

The first three plots represent the Validation Curves vs. C and vs. γ for the case of the 10 folds, while the other hyper-parameter is fixed. The



(a) Plot of the Real Dataset projected onto the 2 principal components extracted with PCA. In red the contaminated samples, in green the uncontaminated/free ones.



(b) Plot of the Real Dataset projected onto the 3 principal components extracted with PCA. In red the contaminated samples, in green the uncontaminated/free ones.

Figure 4.6: 2-D and 3-D plots of the Real Dataset.

selection of the optimal hyper-parameters C and γ is done by looking at the accuracies of these graphs.

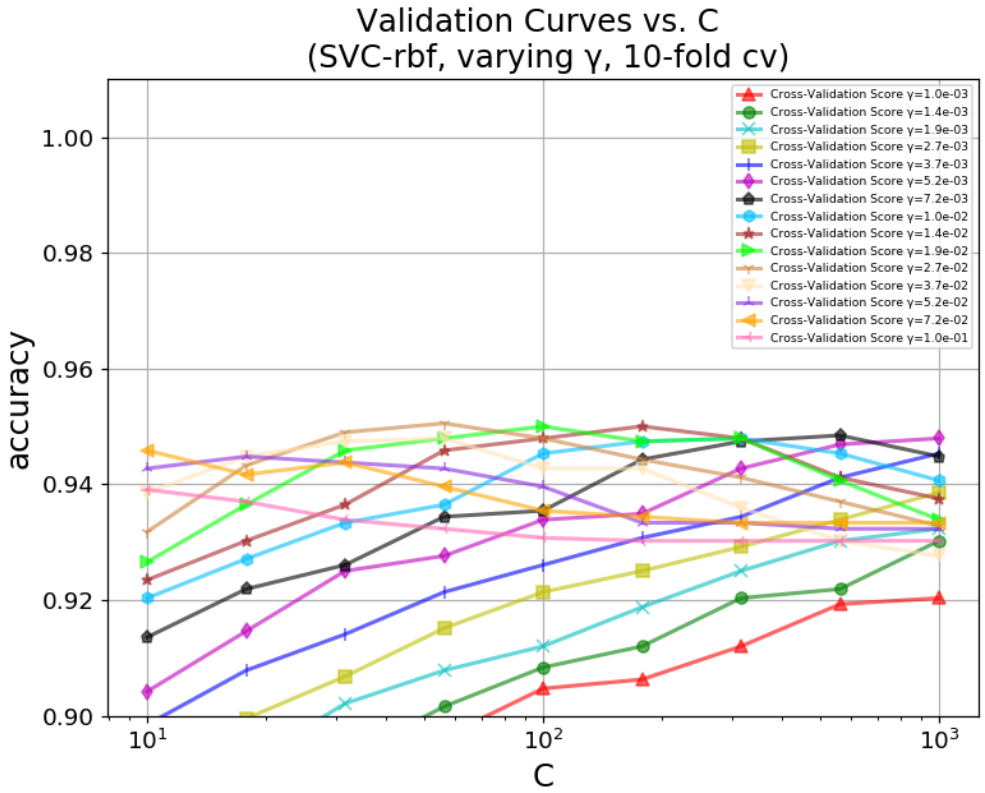


Figure 4.7: Validation Curve vs. C for the SVMs trained on the Real Dataset with the fine grid reported in Table 4.3.

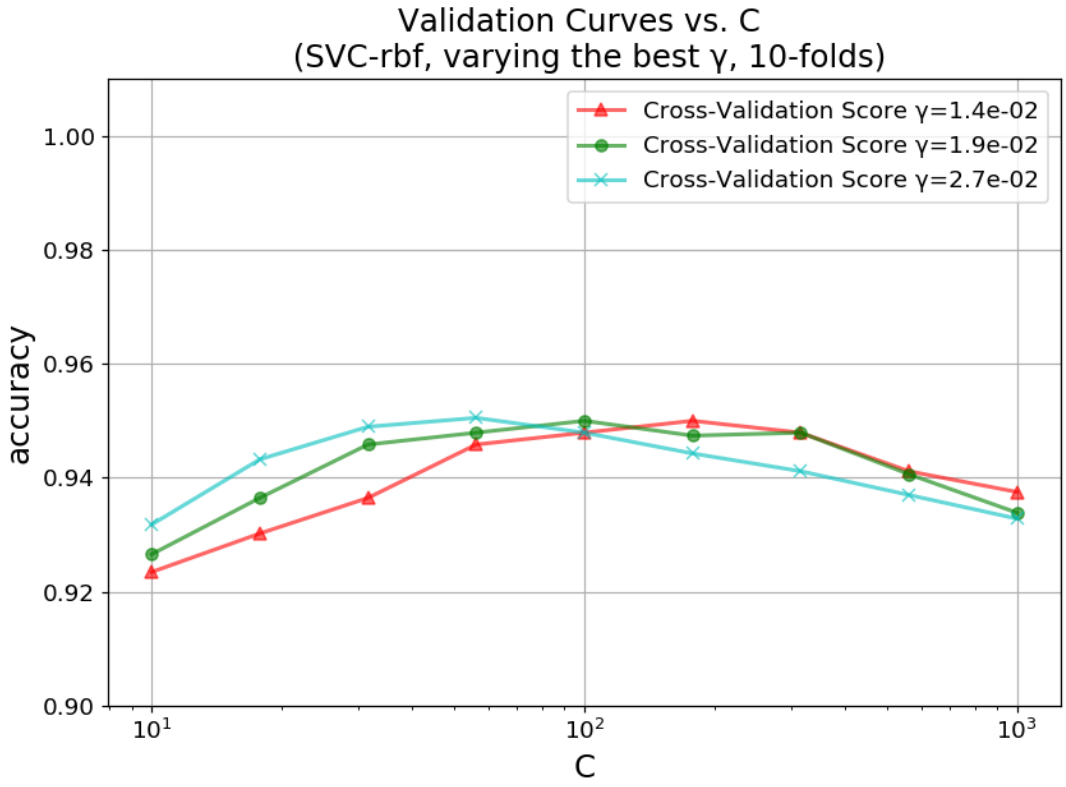


Figure 4.8: Validation Curve vs. C for the SVMs trained on the Real Dataset with the fine grid reported in Table 4.3. The three curves that give the maximum 10-fold CV accuracy of 95.052% are kept with respect to Figure 4.7 for a better visualization.

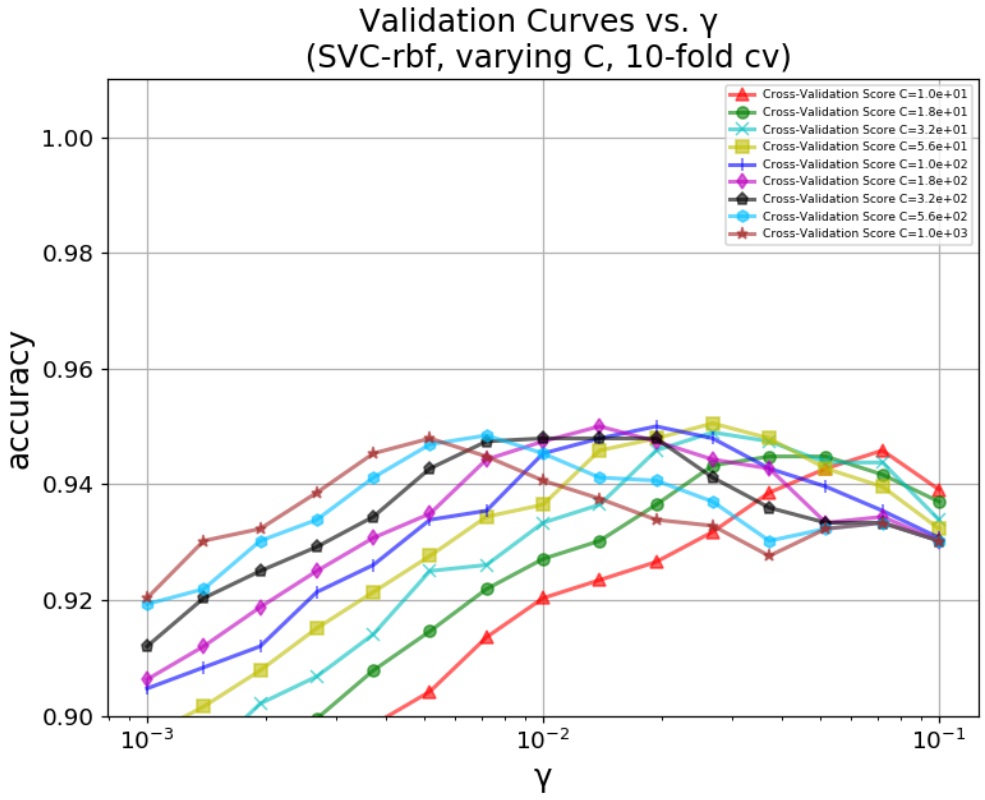


Figure 4.9: Validation Curve vs. γ for the SVMs trained on the Real Dataset with the fine grid reported in Table 4.3.

The next three plots, in Figure 4.10, are the Validation Curves of the three best SVMs for the case of 10 folds. They display the training and validation accuracies vs. C, while γ is fixed. What can be observed is that all the solutions, those marked with the red cross, are at the limit of overfitting because the training and the validation accuracies are both high, but for larger values of C they diverge. So all the solutions are working well and again this confirms that Grid-Search is a good method for finding the best values.

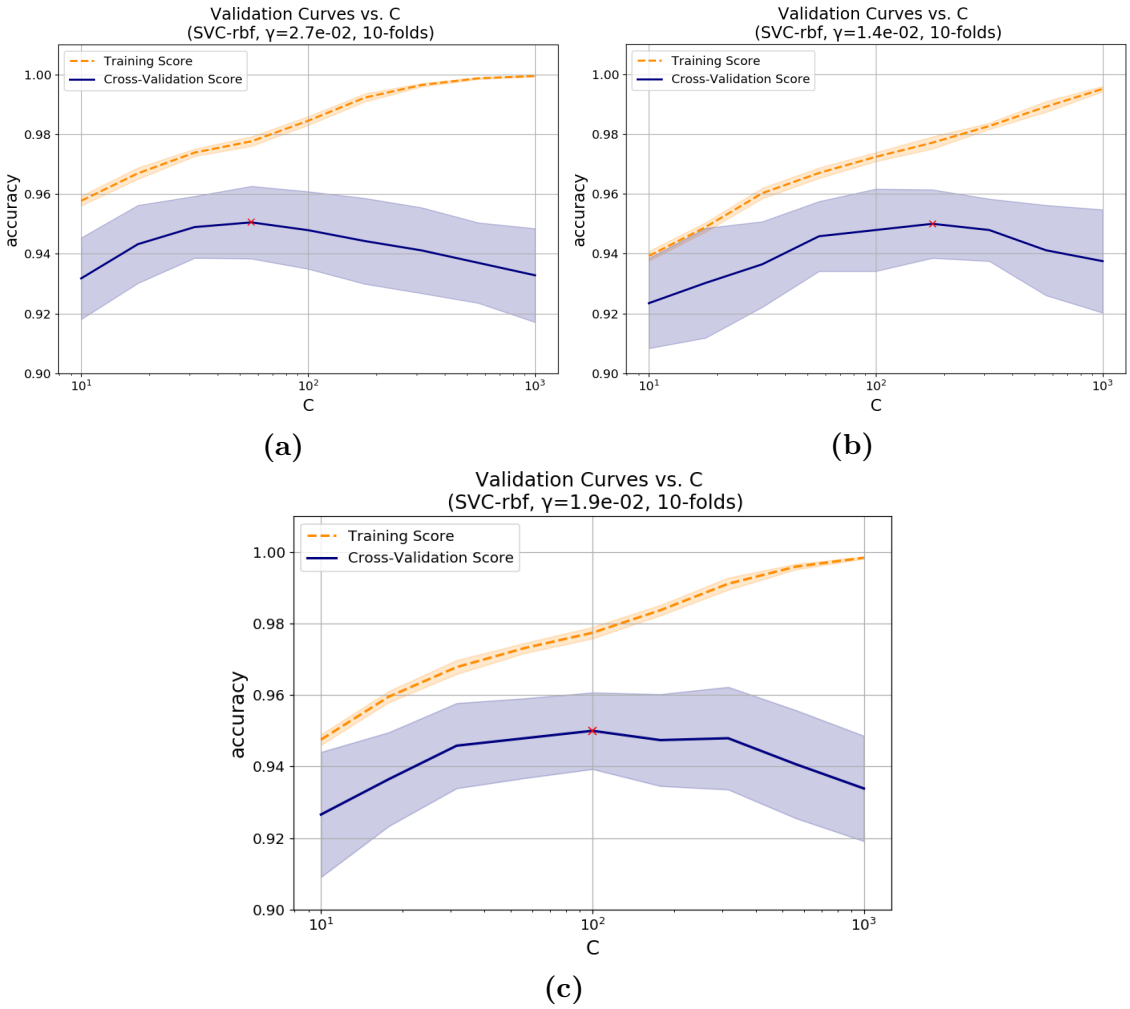


Figure 4.10: "Classical" Validation Curves vs. C of the three SVMs that give the maximum 10-fold CV accuracy of 95.052%, trained on the Real Dataset with the fine grid reported in Table 4.3.

To conclude, the last plots show the Learning Curves of the same three best SVMs. They are in Figure 4.11. From all the graphs, it can be noticed the models have a low bias because the training accuracy is very high, though not at 1.0, and low variance because the training and validation accuracies are getting closer and closer. On the contrary of the synthetic case, these classifiers will benefit the addition of new training data and the actual maximum validation accuracy of 95.052% could increase of around 1-2%.

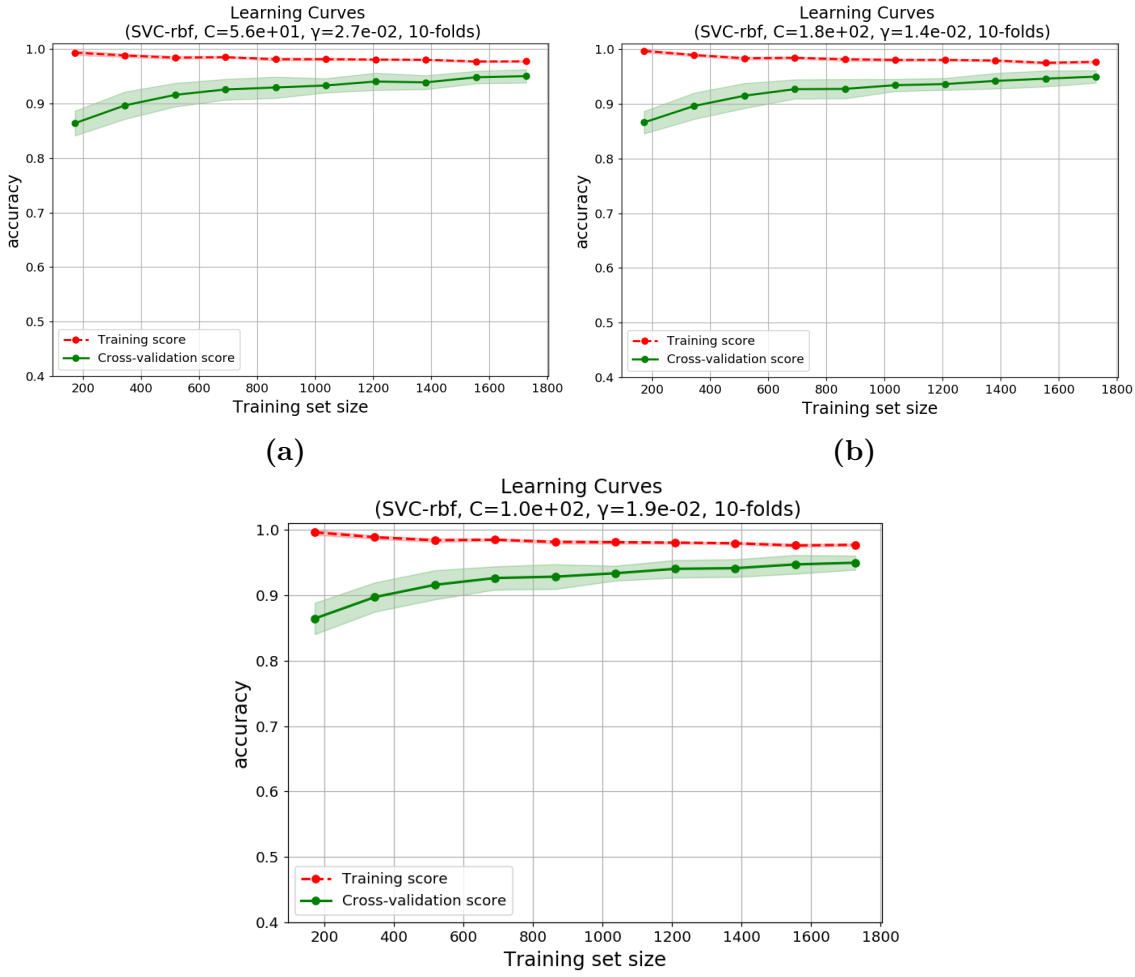


Figure 4.11: Learning Curves of the three SVMs that give the maximum 10-fold CV accuracy of 95.052%, trained on the Real Dataset with the fine grid reported in Table 4.3.

4.3.2 MLPs training and real candidate models

The MLPs are trained on the Real Dataset with the identical procedure discussed in Paragraph 3.3.2. For the Real Dataset three MLPs are trained, respectively with 1, 2, and 3 hidden layers. This time the input layer has 30 neurons because of the number of features of the Real Dataset. The rest of the architecture is still composed by 1, 2 or 3 hidden layers, and an sigmoid output neuron. The loose and the fine grids are reported in Table 4.4, where the second grid is a more accurate version of the first based on the accuracy results of the latter. The constant and tuned parameters of the previous

time continue to be the same, a part the Activation Functions, the Weight Initializers and the Batch-size. Indeed:

- together with Relu and Selu, Tanh is added to the grid of Activation Functions because it demonstrates low training and validation losses when the three kinds of MLPs are trained with the benchmark of Paragraph 3.3.2. The Tanh behavior is visible in Figure 4.12, 4.13, and 4.14;
- it is decided to add Glorot Normal to the Weight Initializers because it is one of the most famous initialization;
- the Batch-size is not fixed at 50 anymore, but it can be 10, 50 or 100, because it was discovered it impacts the regularization of the model: the more it is high, the highest is the regularization, the less the training time.

Grid Type	# Folds	Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Activ. Grid	Weight Initializer Grid	Weight Reg. Param. Grid	Dropout Rate Grid	Batch-size
loose	5	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 4096]			[Relu, Selu, Tanh]	[He, Lecun, Glorot] Normal	[0.0, 0.0001, 0.001, 0.01, 0.1]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55]	[10, 50, 100]
fine	5	[24, 32, 40, 48, 104, 112, 120, 128, 136, 144, 152, 192, 224, 232, 240, 248, 256, 264, 272, 280, 288]			[Relu]	[Lecun, Glorot] Normal	[0.0, 0.0001, 0.001]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55]	[10]
loose	5	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]		[Relu, Selu, Tanh]	[He, Lecun, Glorot] Normal	[0.0, 0.0001, 0.001, 0.01, 0.1]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55]	[10, 50, 100]
fine	5	[32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]		[Relu, Selu]	[Lecun, Glorot] Normal	[0.0, 0.0001, 0.001]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55]	[10, 50, 100]
loose	5	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[Relu, Selu, Tanh]	[He, Lecun, Glorot] Normal	[0.0, 0.0001, 0.001, 0.01, 0.1]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55]	[10, 50, 100]
fine	5	[32, 64, 128, 256, 512, 1024]	[32, 64, 128, 256, 512, 1024]	[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]	[Selu]	[He, Lecun, Glorot] Normal	[0.0, 0.0001, 0.001]	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.55]	[10]

Table 4.4: Grids for the three types of MLP (1, 2 and 3 hidden layers) for the Real Dataset. The fine grids come from the results of the loose grids, reported in Table 4.5, 4.6 and 4.7.

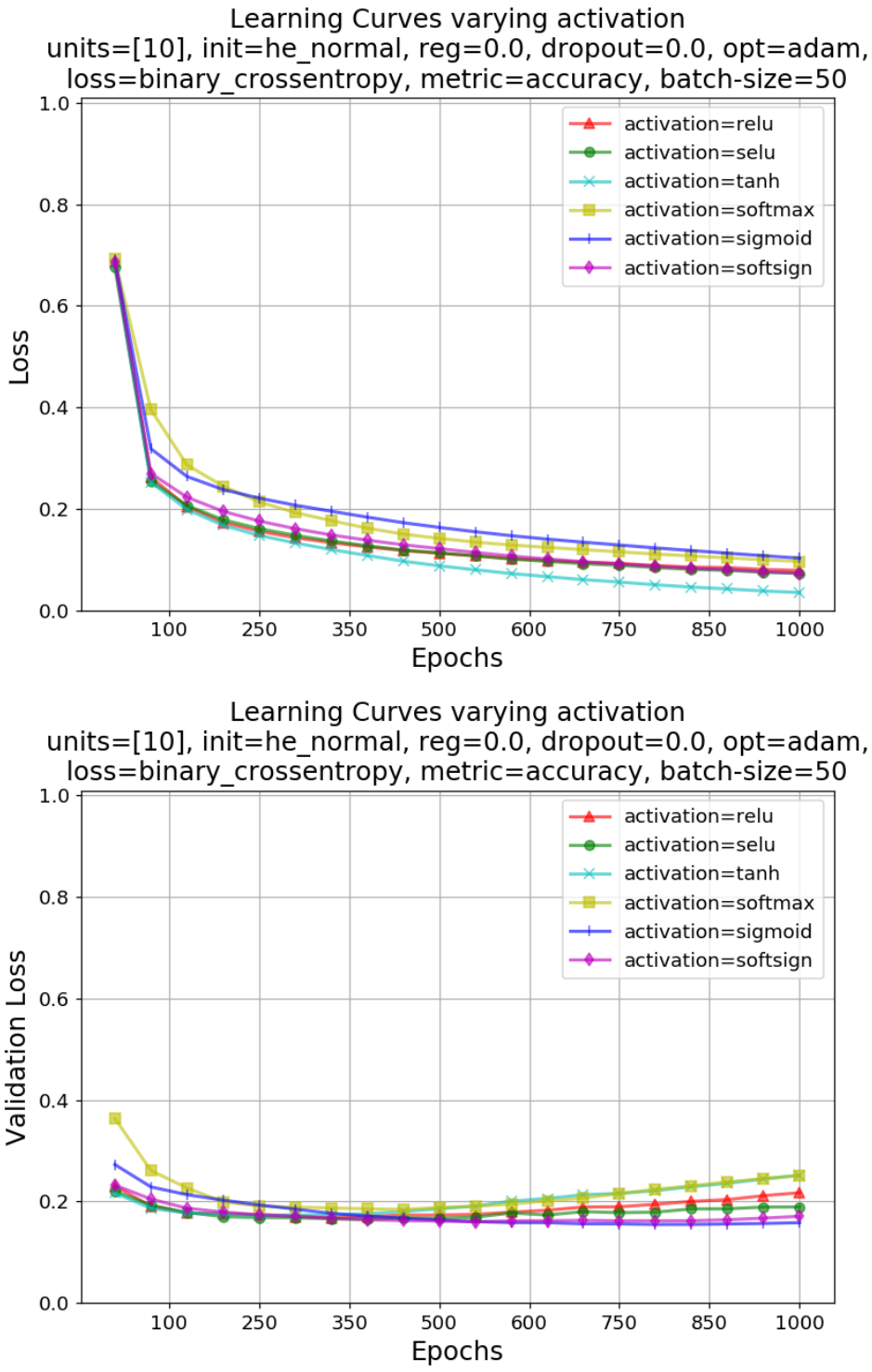


Figure 4.12: 1 hidden layer MLP benchmark for selecting the Activation Functions to use in the Bayesian Optimization grid for the Real Dataset.

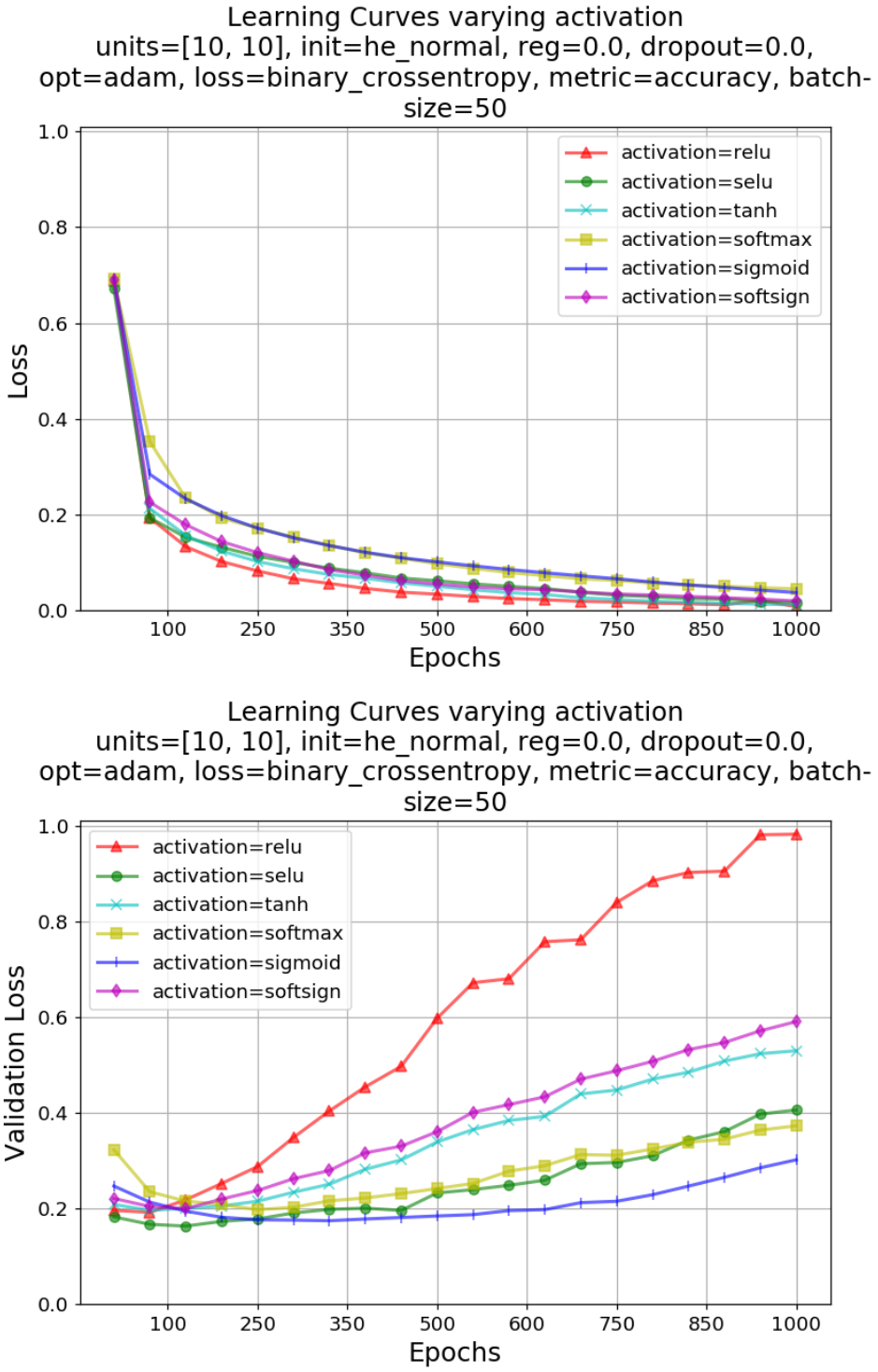


Figure 4.13: 2 hidden layers MLP benchmark for selecting the Activation Functions to use in the Bayesian Optimization grid for the Real Dataset.

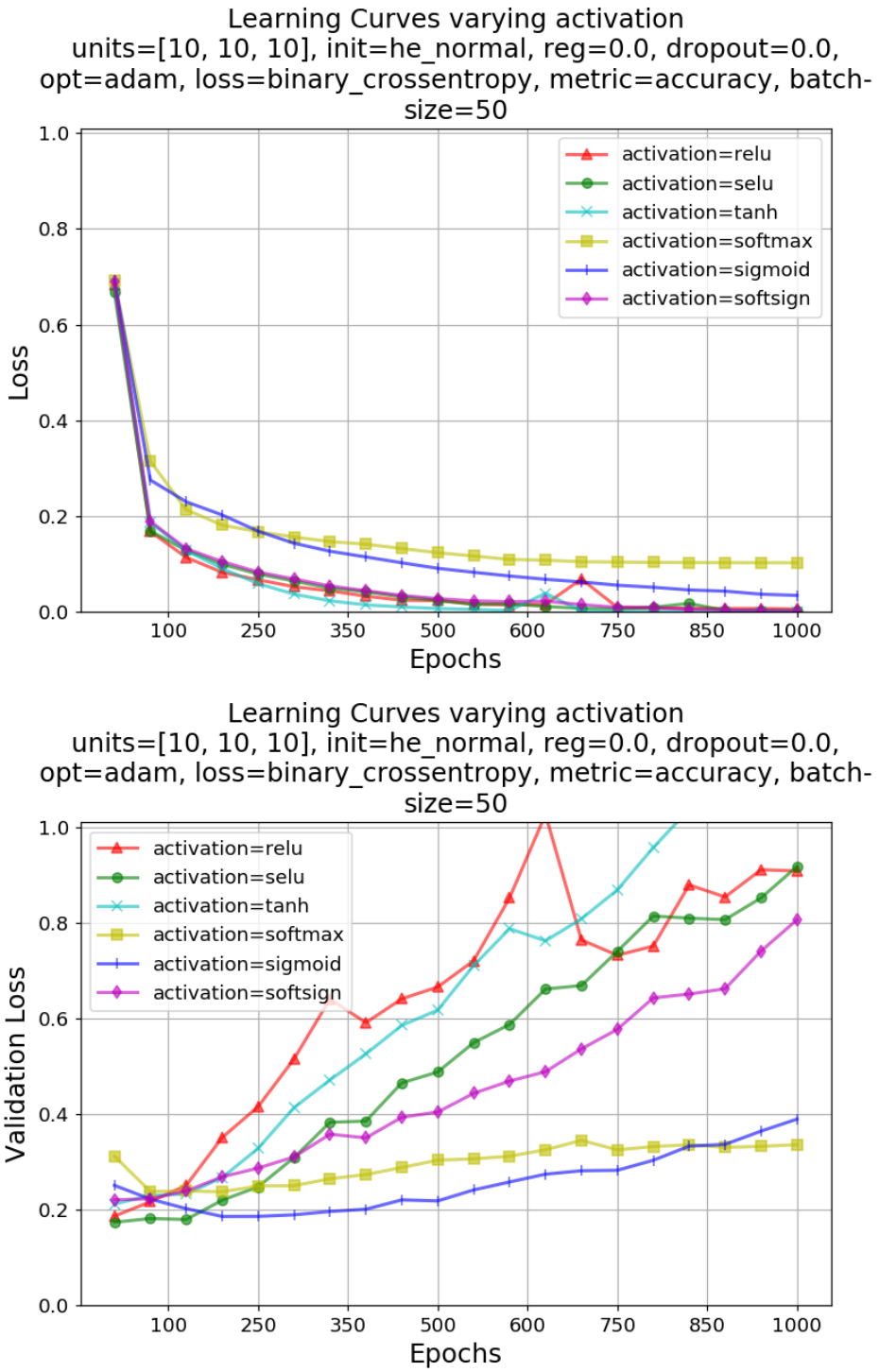


Figure 4.14: 3 hidden layers MLP benchmark for selecting the Activation Functions to use in the Bayesian Optimization grid for the Real Dataset.

The highest training results of the MLPs on the Real Dataset are inserted in three different tables: Table 4.5 for 1 hidden layer, Table 4.6 for 2 hidden layers, and Table 4.7 for 3 hidden layers. As can be seen, many classifiers overcome 94% of 5-fold CV accuracy. Their hyper-parameters will be used in the next section to the create the final MLPs and to test them.

Grid Type	Best Units per layer Hidd. Lay. 1	Hidd. Lay. 2	Hidd. Lay. 3	Best Activ.	Best Weight Init.	Best Weight Reg.	Best Drop-out Rate	Best Batch-size	Best 5-fold CV Acc. (%)	Best Val. Loss (%)
Loose	others								< 93.819	> 0.174
Fine	112			Relu	Lecun Normal	0	0.5	10	93.819	0.174
	192			Relu	Lecun Normal	0.0001	0.5	10	93.819	0.198
	224			Relu	Glorot Normal	0.0001	0.55	10	94.167	0.190
	others								< 93.819	> 0.174

Table 4.5: The most valuable training results for the Real Dataset for a MLP with 1 hidden layer.

Grid Type	Best Units per layer Hidd. Layer 1	Hidd. Layer 2	Hidd. Layer 3	Best Activ.	Best Weight Init.	Best Weight Reg.	Best Drop-out Rate	Best Batch-size	Best 5-fold CV Acc. (%)	Best Val. Loss (%)
Loose	others								< 94.097	> 0.237
Fine	32	128		Selu	Lecun Normal	0	0.1	10	94.236	0.163
	128	256		Relu	Glorot Normal	0.001	0.5	10	94.097	0.237
	others								< 94.097	> 0.237

Table 4.6: The most valuable training results for the Real Dataset for a MLP with 2 hidden layers.

Grid Type	Best Units per layer Hidd. Layer 1	Hidd. Layer 2	Hidd. Layer 3	Best Activ.	Best Weight Init.	Best Weight Reg.	Best Drop-out Rate	Best Batch-size	Best 5-fold CV Acc. (%)	Best Val. Loss (%)
Loose	512	64	128	Selu	He Normal	0	0.3	10	94.444	0.159
	64	256	128	Selu	He Normal	0.0001	0.1	10	94.306	0.227
	512	64	128	Selu	He Normal	0	0.4	10	94.097	0.162
	others								< 94.097	> 0.163
Fine	64	256	64	Selu	He Normal	0	0.3	10	94.375	0.163
	others								< 94.097	> 0.163

Table 4.7: The most valuable training results for the Real Dataset for a MLP with 3 hidden layers.

4.4 Testing, performance evaluations and best real models

In this section the candidate classifiers of the previous two paragraphs are tested on the held-out 20% of the Real Dataset and their performances are shown with the usual metrics. It is important to underline that the performance metrics of two classifiers trained on different datasets can't be compared. So the results of Paragraph 4.4.1 have to be compared to those of Paragraph 4.4.2 only.

4.4.1 SVMs testing, performance and best real model

The results of the tests of the most performing SVMs on the Real Test Set are reported in Table 4.8. The difference compared to the synthetic case is the calculus of the Error Rate because its denominator is now 480. The winner is the SVM with (C, γ) = (5.6E+01, 2.7E-02) with 29 samples mispredicted over 480, which corresponds to an error rate of 6.042%. It confirms the primacy conquered during the training stage.

Best (C, γ)	**Best CV Training Acc. (%)**	**Test Acc.**	**Errors**	**Error Rate (%)**
(5.6E+01, 2.7E-02)	**95.052**	**93.958**	**29**	**6.042**
(1.8E+02, 1.4E-02)	95.052	93.333	32	6.667
(1.0E+02, 1.9E-02)	95.052	93.750	31	6.458

Table 4.8: Summary of the test results of the SVMs of Table 4.3. The best SVM is in bold.

The performance metrics of the SVM with (C, γ) = (5.6E+01, 2.7E-02) are given in the subsequent figures.

Figure 4.15 is the Confusion Matrix. It tells that precision is at 0.969 and recall is at 0.908. Of course, it would be better to have a lower number of False Negatives than False Positives because it is safer to discard a jar which could be a potential hazard (although it is not), rather than let it pass labelled as "free".

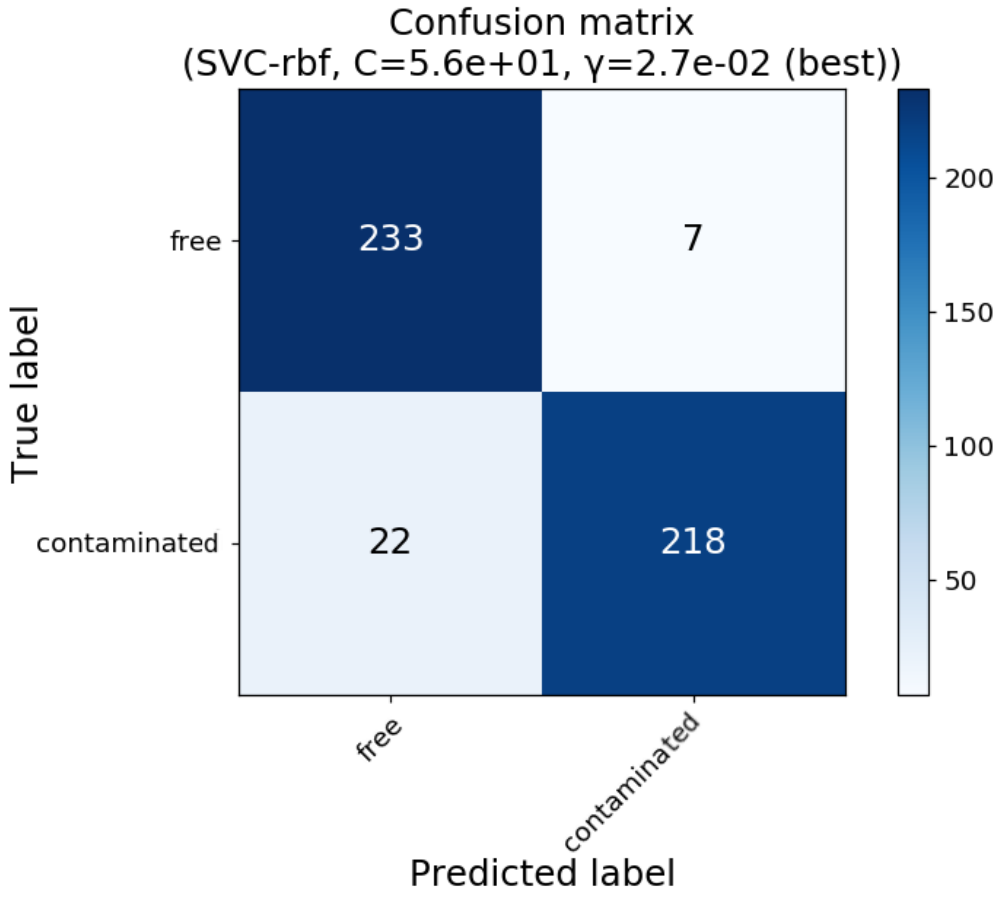


Figure 4.15: Confusion Matrix of the best found SVM, the one with (C, γ) = (5.6E+01, 2.7E-02), calculated with the Real Test Set.

In Figure 4.16 precision and recall scores are plotted versus the decision threshold of the SVM. From this graph it could be noticed that the number of False Negatives could be reduced until 0 with threshold adjustment, by moving the decision threshold of the SVM on the left until -1.87. Despite the problem of high False Negatives is solved, the occurrences of False Positives, and so the precision (the solid line in light blue in the plot) would raise, leading to a substantial increase of the discarded jars. Therefore, if threshold adjustment is employed in future works, it has to be carefully controlled to minimize the sum of False Negatives and False Positives together [43].

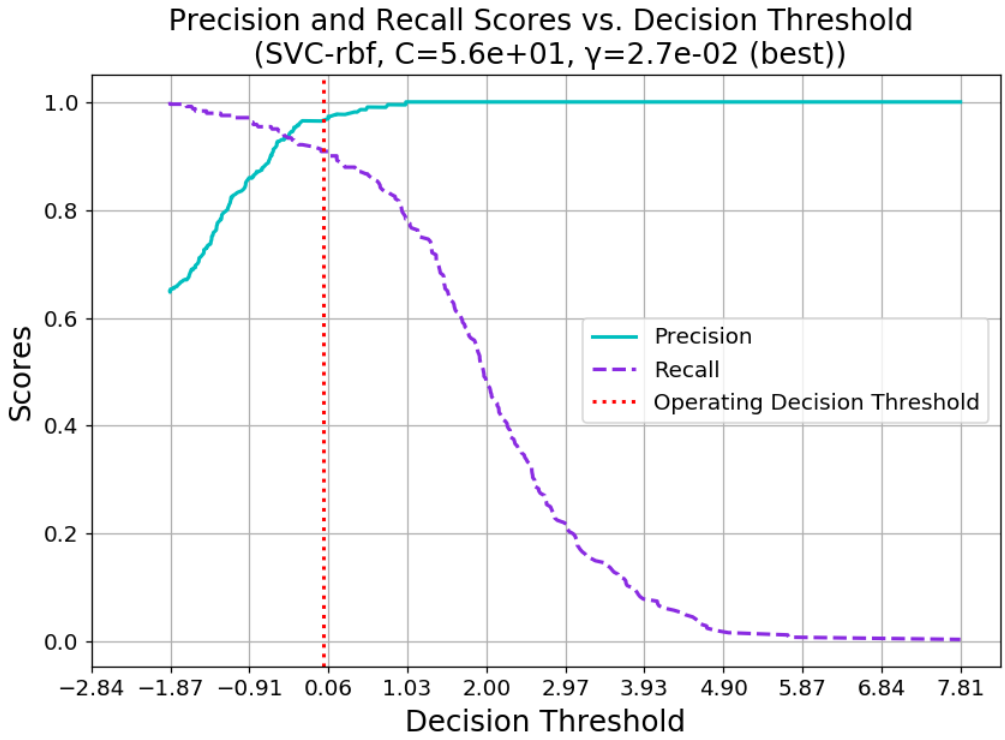


Figure 4.16: Precision and recall scores vs. decision threshold of the best found SVM, the one with $(C, \gamma) = (5.6\text{E+}01, 2.7\text{E-}02)$, calculated with the Real Test Set.

The last figure (4.17) is the ROC Curve. Although the operating point of this classifier is not at coordinates (0, 1), it is still very near to it and the AUC score is at 0.982. The classifier is working fine.

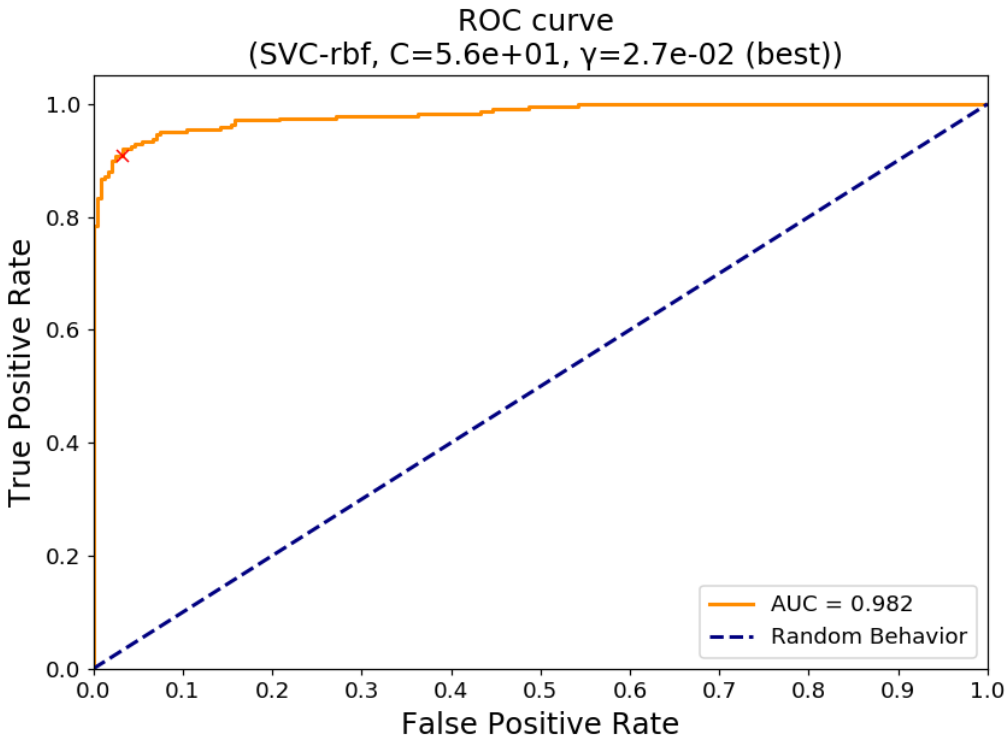


Figure 4.17: ROC Curve of the best found SVM, the one with (C, γ) = (5.6E+01, 2.7E-02), calculated with the Real Test Set.

Also this time 30-fold nested cross-validation (30-fold in both the inner and outer loops) is performed for this SVM on the whole Real Dataset (not split in 80%-20%). In Table 4.9 are reported the results. Among the 30 folds, many couples (C, γ) are found by Grid-Search. Here it is decided to keep those with the highest number of occurrences, that is those that were selected more often as best classifiers by Grid-Search in the inner loop. They are two: the first has (C, γ) = (5.6E+01, 3.7E-02) and gets (95.714 ± 0.976)% with a 95% confidence level; the second has (C, γ) = (5.6E+01, 7.2E-02) and reaches (94.000 ± 0.815)% with a 95% confidence level. The training grids are taken from the 10-fold non-nested CV case of Table 4.3. Differently from the synthetic case where the solution that gave the highest nested accuracy had different hyper-parameters from that discovered during the training phase, now both solutions have C identical to that of the 10-fold non-nested CV case ($C_{nested} = C_{non-nested}$ = 5.6E+01), while γ is a bit different but with the same order of magnitude (γ_{nested_1} = 3.7E-02, γ_{nested_2} = 7.2E-02, $\gamma_{non-nested}$ = 2.7E-02). Even if the final results of nested and non-nested methods shouldn't be compared, it is interesting too see they arrive

to almost the same best solution, in this case.

Occurrences	**Best** (C, γ)	**Best CV Training Acc. (%)**	**95% Confindence Interval**	**Worst Case**
7	**(5.6E+01, 3.7E-02)**	**95.714**	**0.976**	**94.738**
5	(5.6E+01, 7.2E-02)	94.000	0.815	93.185
< 5	others	< 94.000		

Table 4.9: Nested-CV results of the classifier trained on the Real Dataset. The best nested SVM is in bold.

As last consideration, it is nice to understand which types of contaminants are mispredicted to improve the classification capabilities of the *MIT-Food prototype* in future. From Table 4.10 it is possible to see that the strongest contribute of error comes from the triangular plastic fragment (Figure 4.5f, 4.5e, and 4.5g). In fact, in this case the outcome of the classifier behaves like a coin toss, with an accuracy of 51.220%. However, if we image to exclude this case for a moment, the resulting error rate on the test set would be 2.050% instead of 6.042%. The reasons why this small triangular plastic fragment is mispredicted are not due to its small dimensions, as one could think. In fact, the small plastic sphere, which has a diameter of 3 mm against the triangular plastic that has dimensions 8 mm x 6 mm x 1 mm, is correctly recognized. A possible interpretation to this may be the following. Since the triangular piece of plastic is floating onto the oil surface, the *MWI prototype system* misinterprets the signal as coming from a position outside the volume where oil is present because the dielectric contrast between air-plastic interface is greater than the plastic-oil one. This is for sure an aspect that needs an additional investigation in future works.

Type of Sample	Occurrences in Test Set	Errors	Error Rate (%)
Free = No contaminant	240	7	2.917
Metal sphere	45	1	2.222
Glass fragment	43	1	2.326
Big plastic sphere	37	0	0
Small plastic sphere	35	0	0
Triangular plastic fragment	41	20	48.780
Cap shape plastic	39	0	0
Sum	480	29	6.042
Sum without triangular plastic fragment	439	9	2.050

Table 4.10: Types of samples in the Real Test Set and their contribution to the final Error Rate. The number of occurrences of the various types of contaminants in the Real Test Set are not uniform because the test samples were selected randomly from the shuffled Real Dataset.

4.4.2 MLPs testing, performance and best real model

Exactly as happened before for the most promising synthetic candidates, new MLPs are trained from scratch with the same hyper-parameters that gave the highest 5-fold CV accuracy in Paragraph 4.3.2. They are manually trained on the 75% of the Real Training Set, corresponding to 1920 training samples, before their evaluation on the Real Test Set. The remaining 25% of the initial Real Training Set is used for validation purposes, i.e. to monitor the learning process and to prevent overfitting. The value 25% is adopted arbitrarily to create a validation set of 480 samples as large as the Real Test Set. The resulting models are also saved to the disk for a possible subsequent hardware implementation.

The tests which are carried out on the brand new trained MLPs candidates are reported in Table 4.11 for 1 hidden layer, Table 4.12 for 2 hidden layers, and Table 4.13 for 3 hidden layers.

The most accurate MLPs are summarized in Table 4.14. The classifier which outperforms the others is in bold: it is a 2 hidden layers MLP with [128, 256] Relu Units which has an error rate of 6.042%, mispredicting 29 out of 480 test samples.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Best Batch-size	Epochs	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	Errors	Error Rate (%)
112			Relu	Lecun Normal	0	0.5	10	54	95.208	0.142	91.875	39	8.125
192			Relu	Lecun Normal	0.0001	0.5	10	76	95.208	0.163	93.542	31	6.458
224			**Relu**	Glorot Normal	0.0001	0.55	10	90	**96.042**	**0.154**	**93.750**	**30**	**6.250**

Table 4.11: Test results on the Real Test Set of the manually trained MLPs with 1 hidden layer. The best MLP is in bold.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Best Batch-size	Epochs	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	Errors	Error Rate (%)
32	128		Selu	Lecun Normal	0	0.1	10	98	95.000	0.146	93.333	32	6.667
128	**256**		**Relu**	Glorot Normal	0.001	0.5	10	184	**95.833**	**0.199**	**93.958**	**29**	**6.042**

Table 4.12: Test results on the Real Test Set of the manually trained MLPs with 2 hidden layers. The best MLPs are in bold.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Weight Init.	Best Weight Reg.	Best Dropout Rate	Best Batch-size	Epochs	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	Errors	Error Rate (%)
512	**64**	**128**	**Selu**	He Normal	0	0.3	10	38	**96.250**	**0.142**	**93.542**	**31**	**6.458**
64	256	128	Selu	He Normal	0.0001	0.1	10	18	95.208	0.216	91.250	42	8.750
512	64	128	Selu	He Normal	0	0.4	10	42	95.833	0.155	93.125	33	6.875
64	256	64	Selu	He Normal	0	0.3	10	40	95.625	0.138	92.292	37	7.708

Table 4.13: Test results on the Real Test Set of the manually trained MLPs with 3 hidden layers. The best MLP is in bold.

Best Units per layer Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Best Activation	Best Val. Acc. (%)	Best Val. Loss (%)	Test Acc.	Errors	Error Rate (%)
224			Relu	96.042	0.15381	93.750	30	6.250
128	**256**		**Relu**	**95.833**	**0.199**	**93.958**	**29**	**6.042**
512	64	128	Selu	96.250	0.142	93.542	31	6.458

Table 4.14: Summary of the test results of the MLPs in Table 4.11, 4.12 and 4.13. The best MLP is in bold.

The Loss Curves and the Learning Curves of the manual training of the best MLP are reported in Figure 4.18. The Training Loss decreases rapidly for the first 50 epochs, then diminishes slowly and remains above 0.1. This means the model has difficulty in learning from the training data. On the other hand, the Validation Loss stabilizes immediately, just after 40 epochs. It doesn't increase, but it doesn't improve more than about 0.2. This behavior would lead to a model with high variance in the long term, but the model is stopped and saved at epoch 184, where the maximum of Validation Accuracy of 95.833% is reached. It also corresponds to one of the minima of the Validation Loss, resulting in a small discrepancy with the Training Loss, and so a low variance. Probably, continuing the training for more epochs, the Training Loss will continue to decrease, while the Validation Loss will stabilize or will increase, overfitting the model. The same reasonings can be done for the Learning Curves which are an equivalent version of the Loss Curves.

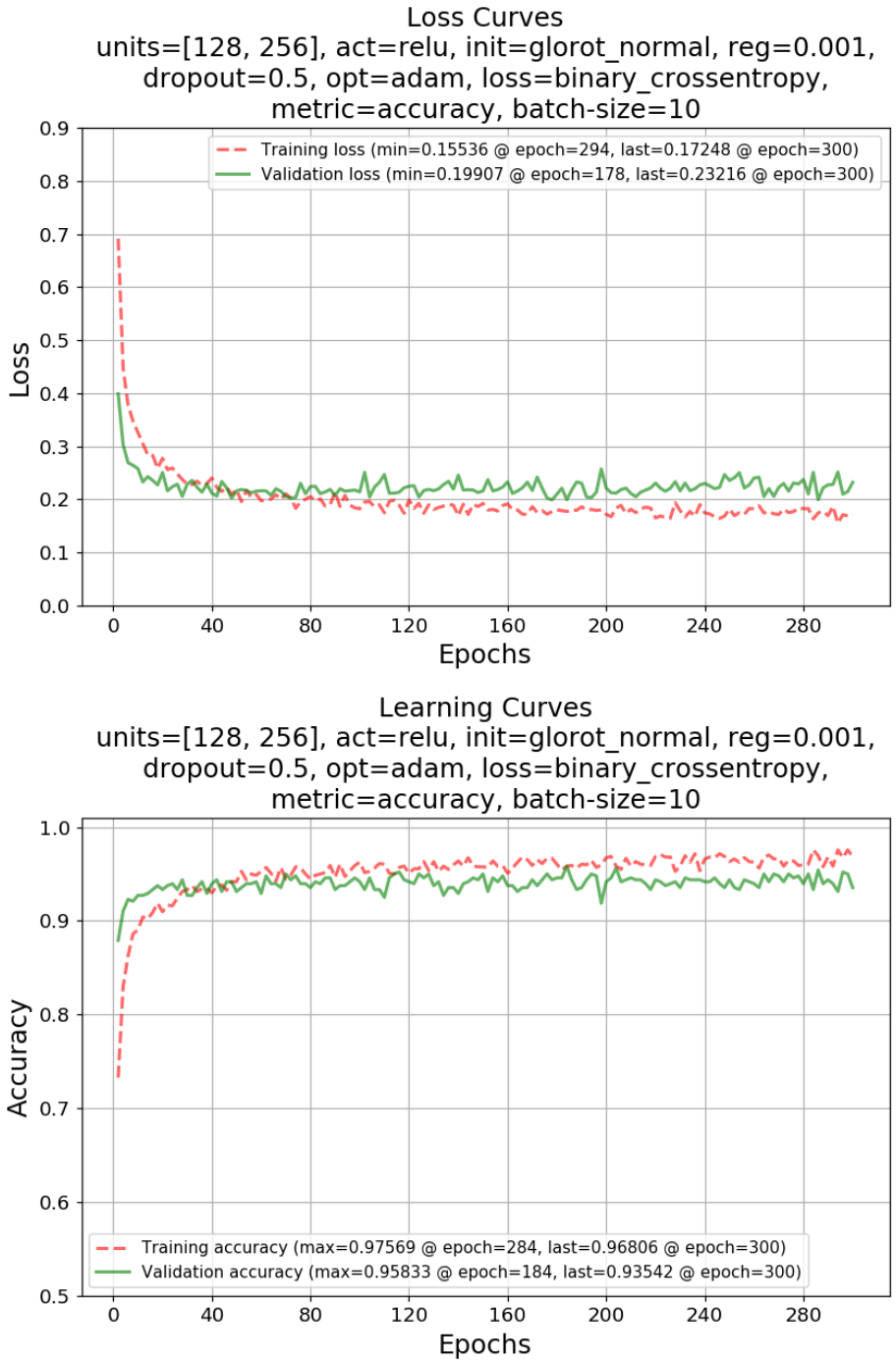


Figure 4.18: Loss Curves and the Learning Curves of the best found MLP with 2 hidden layers and [128, 256] Relu Units, trained with 75% of the Real Training Set and validated on the remaining 25%.

The performance of the selected MLP with 2 hidden layers and [128, 256] Relu Units are given with the following plots.

Staring with the Confusion Matrix in Figure 4.19 it can be seen that precision is at 0.965 and recall is at 0.912. To make a comparison with the Confusion Matrix of the final SVM of Figure 4.15 where precision is at 0.969 and recall 0.908, the MLP has 1 misprediction less in the False Negatives and 1 more in the False Positives. Overall, the total number of mispredictions is 29 for both.

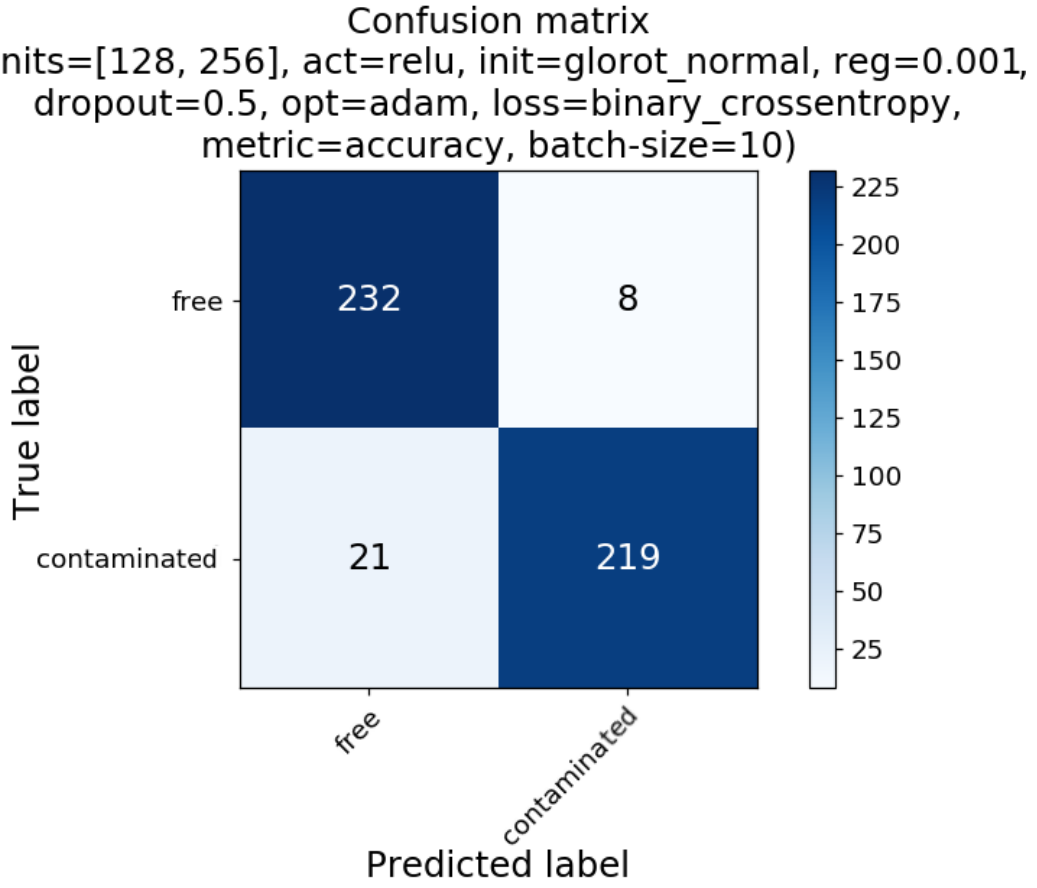


Figure 4.19: Confusion Matrix of the best found MLP with 2 hidden layers and [128, 256] Relu Units, calculated with the Real Test Set.

In Figure 4.20 precision and recall scores are plotted versus the decision threshold of the MLP. The decision threshold is still at 0.5 and threshold adjustment could improve recall, paying attention to possible False Positives increments.

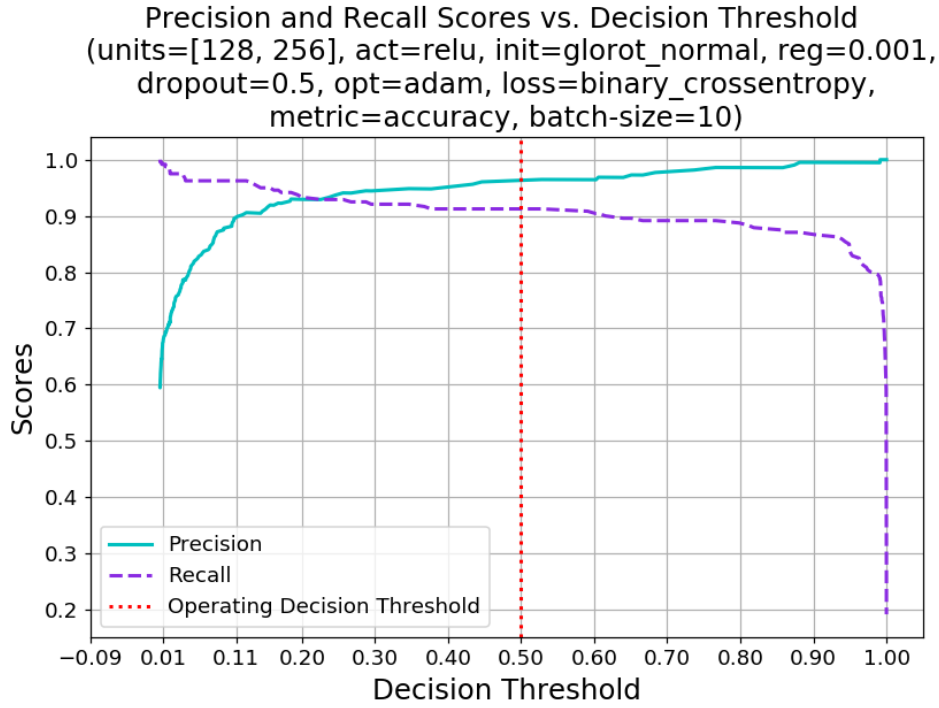


Figure 4.20: Precision and recall scores vs. decision threshold of the best found MLP with 2 hidden layers and [128, 256] Relu Units, calculated with the Real Test Set.

Finally, the ROC Curve is in Figure 4.21. The operating point of this MLP is near the point (0, 1) and its AUC is 0.980. Instead, the AUC of the final SVM was 0.982.

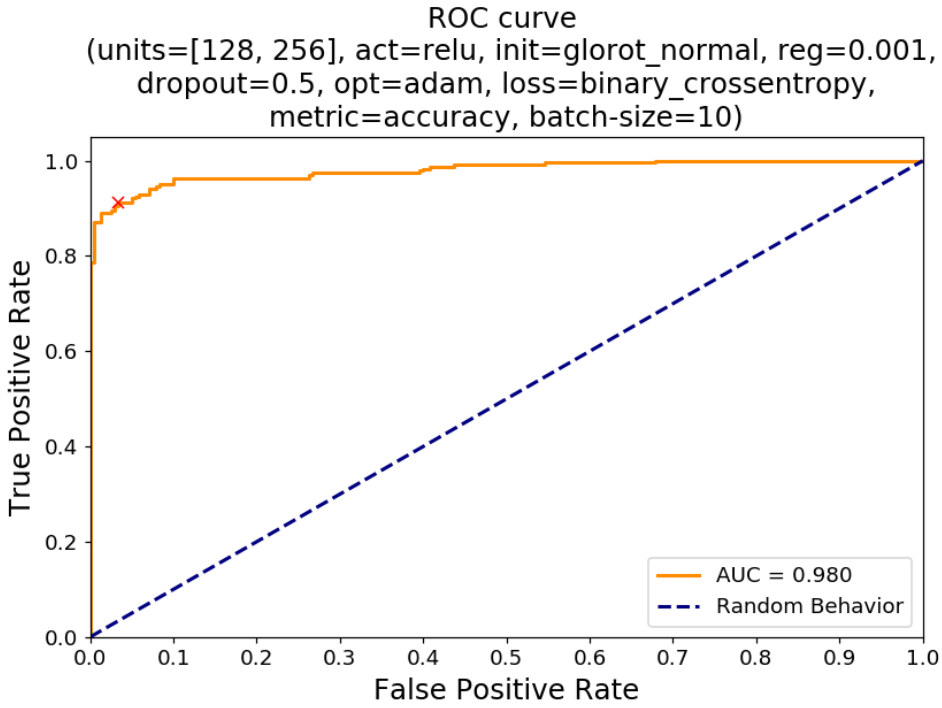


Figure 4.21: ROC Curve of the best found MLP with 2 hidden layers and [128, 256] Relu Units, calculated with the Real Test Set.

As already stated in Paragraph 3.4.2, nested cross-validation is not employed for its extremely long training time. 5-fold non-nested CV proved to give good results for this application with a relatively fast training.

At the end, Table 4.15 illustrates which are the types of contaminants that are mispredicted. Once again, the triangular plastic fragment (Figure 4.5f, 4.5e, and 4.5g) is the cause of most of the errors (21). Indeed, this MLP makes the 51.229% of errors for this kind of hazards: a very poor result and still useless as the 48.780% of the final SVM. On the other hand, this MLP predicts correctly all the other test samples of the remaining types of intrusions. If the errors due to the triangular plastic fragment are discarded, the resulting Error Rate on the test set would decrease at 1.882%, instead of the overall 6.042%. It is also less than the same Error Rate calculated on the final SVM which was 2.050%.

Type of Sample	Occurrences in Test Set	Errors	Error Rate (%)
Free = No contaminant	240	8	3.333
Metal sphere	45	0	0
Glass fragment	43	0	0
Big plastic sphere	37	0	0
Small plastic sphere	35	0	0
Triangular plastic fragment	41	21	51.220
Cap shape plastic	39	0	0
Sum	480	29	6.042
Sum without triangular plastic fragment	439	8	1.822

Table 4.15: Types of samples in the Real Test Set and their contribution to the final Error Rate.

To conclude, this MLP with 2 hidden layers and [128, 256] Relu Units is chosen for the last step of this thesis, which is the hardware implementation. The reasons that support its candidacy against the final SVM with $(C, \gamma) =$ (5.6E+01, 2.7E-02) are:

- it has a lower number of False Negatives which are more important than False Positives because this system has to impede a contaminated jar passes its control;
- it is able to predict all the intrusions with no errors, except the triangular plastic fragment;
- its hardware realization is faster because of its straightforward architecture;
- it has the an Error Rate of 6.042% on the test set equal to the SVM.

Chapter 5

Hardware acceleration

This chapter talks about all the steps involved in the developing of an hardware accelerator for the best MLP discovered in Chapter 4, whose architecture is [30, 128, 256, 1]. In short, they consist in: Keras model conversion in a synthesizable C/C++ code, High Level Synthesis (HLS) with design space exploration, and physical implementation on FPGA for the estimations of latency, utilization of resources, and power.

5.1 Model conversion in a synthesizable code

As already said in Paragraph 4.4.2, during the training of the MLPs on the 75% of the Real Training Set, their architecture and their weights are saved on the disk. These two are the ingredients for the Keras model conversion into a synthesizable C/C++ code. In fact, when the number of neurons in each layer of the network, the number of hidden layers, and the activation functions inside each neuron are known, the MLP is completely defined.

Instead of writing the code manually, the conversion is realized by means of a package called *hls4ml*[1]. It takes the architecture of the MLP as a JSON file, the weights as a HDF5 file, the target FPGA, and automatically generates a synthesizable C++ code (together with a skeleton of the testbench) ready to be imported in Xilinx Vivado tools. However, that code doesn't include the standardization of the inputs, a fundamental step to make the MLP work as expected. So it is edited to add the Standardization Block,

[1] `https://fastmachinelearning.org/hls4ml/`

which standardizes the inputs with the mean and standard deviation values obtained during the preprocessing phase of Paragraph 4.2. In this way, new unseen data undergo the same standardization of the training samples. In Figure 5.2 the additional lines of code inserted in the C++ code for the *Standardization Block* are presented.

- the arrays called "mean" and "stddev" contain the values necessary to compute the standardization (lines 107-108);
- "input1" is the input sample with length "N_INPUT_1_1" = 30 (line 91). It is redirected from the input of the network (line 125) to the input of the Standardization (line 116);
- "input2" is the output of the Standardization Block (lines 115-117) that goes to the input of the network (line 126), always with length 30 (line 113).

The target FPGA for this project is the *Xilinx xc7z020clg484-1* mounted on the *Avnet ZedBoard Zynq-7000 Development Board*, whose picture is in Figure 5.1.

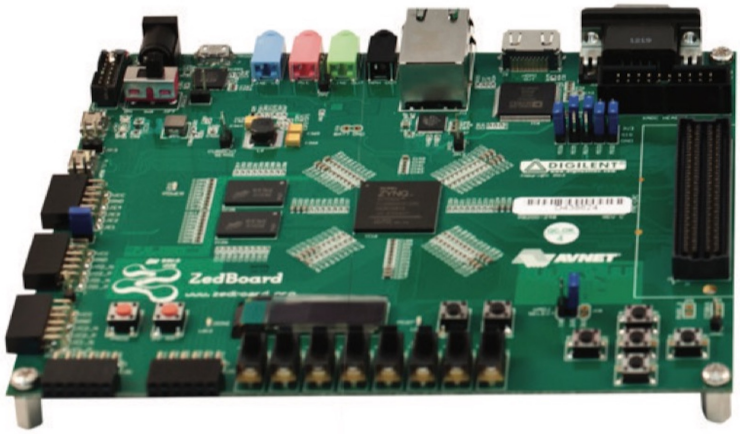


Figure 5.1: Picture of the *Avnet Zedboard Zynq-7000 Development Board*.

```
84
85  #include "MLP-30-128-256-1.h"
86  ...
87  #include "weights/mean.h"
88  #include "weights/stddev.h"
89
90  void MLP-30-128-256-1(
91      input_t input1[N_INPUT_1_1],
92      result_t layer7_out[N_LAYER_6]
93  ){
94
95      ...
96
97      // ************************************
98      // PREPROCESSING - STANDARDIZATION
99      // ************************************
100
101 #ifndef __SYNTHESIS__
102     static bool loaded_weights = false;
103     if (!loaded_weights) {
104
105         ...
106
107         nnet::load_weights_from_txt<model_default_t, 30>(mean, "mean.txt");
108         nnet::load_weights_from_txt<model_default_t, 30>(stddev, "stddev.txt");
109         loaded_weights = true;
110     }
111 #endif
112
113     input_t input2[N_INPUT_1_1];
114
115     Standardization: for (int iscal=0; iscal<N_INPUT_1_1; iscal++) {
116         input2[iscal] = (input1[iscal] - mean[iscal]) / stddev[iscal];
117     }
118
119     // ************************************
120     // NETWORK INSTANTIATION
121     // ************************************
122
123     ...
124
125 //    nnet::dense_latency<input_t, layer2_t, config2>(input1, layer2_out, w2, b2);
126     nnet::dense_latency<input_t, layer2_t, config2>(input2, layer2_out, w2, b2);
127
128     ...
129
130 }
131
```

Figure 5.2: C++ code with the additional lines for the *Standardization Block*.

5.2 Design space exploration

hls4ml allows to configure various settings to automatically custom the code with the desired Vivado HLS directives. They are: clock period, internal fixed-point precision, pipelining, resource reuse, "latency" or "resource" strategy. It is decided to try different combinations of those parameters to explore various kinds of architectures in Vivado HLS. For each solution four values are set for the clock period while maintaining the other settings fixed: 5, 10, 50 and 100 ns. Overall, the goal is always to maintain the correct functionality, the latency under 100 ms, and the resource utilization below the limits of the target FPGA.

All the architectures are required to pass the C simulation that takes the samples of the Real Test Set as stimuli. Around 50 architectures are developed. Those that pass the C simulation and provides interesting results in terms of latency and resource utilization after the High Level Synthesis are reported in Table 5.1. All of them have an internal fixed-point precision of <64, 32>, where 64 is the internal bit-width parallelism and 32 (out of 64) are the fractional bits, and are generated with a "latency" strategy because lower precisions and "resource" strategy prevented the C simulations to pass. The architectures in the table continue with physical implementation in Vivado, which is described in Paragraph 5.3.

Solution	**Target Tck (ns)**	**Latency (ms)**	**BRAMs (%)**	**DSPs (%)**	**FFs (%)**	**LUTs (%)**
1	5	4.611	84	22	14	22
2	10	4.118	84	22	5	10
3	50	7.682	84	14	3	9
4	100	15.364	84	14	3	9
5	5	3.503	85	22	15	27
6	10	2.989	85	22	5	16
7	50	3.953	85	14	3	14
8	100	7.906	85	14	3	14
9	5	0.361	86	80	50	119
10	10	0.602	86	80	36	110
11	50	3.010	86	23	29	105
12	100	6.019	86	23	29	105

Table 5.1: Post-Synthesis results of the most interesting solutions. The resources that exceed the limits are highlighted in red.

5.3 Physical implementation and performance estimations

The physical implementation on the ZedBoard requires that the architectures in Table 5.1 are saved as *Intellectual Property* (IP) *Cores* and imported in Vivado. Then, each IP has to be connected to the *Zynq Processing System* (PS) and to a 64-bit wide *BRAM*. The adopted interface is a 64-bit AXI Interface and the connections are illustrated in Figure 5.3. In this way, more realistic performance can be estimated because the ZedBoard ecosystem is taken into account. At this point the implementation of each solution is carried out until the *Bitstream Generation*. The estimations are reported in Table 5.2.

From the results one can infer that the lowest latency is achieved by solution 6, with 2.997 ms. In addition, that solution is also competitive with the others in terms of resource usage and power consumption. The calculus of the latency is given by the multiplication of the number of clock cycles obtained with the *RTC/C Co-Simulation* and the target clock period T_{ck}. The last solution, instead, has not finished the implementation because it exceeded the limit of hardware resources of the FPGA. These estimations belong to the PL only.

Solution	Target Tck (ns)	Latency (ms)	BRAMs (%)	DSPs (%)	FFs (%)	LUTs (%)	Power (W)
1	TIMING	NOT MET	42	11	18	39	3.055
2	10	4.119	42	11	8	13	1.990
3	50	7.684	41	11	7	17	1.766
4	100	15.367	41	11	7	17	2.653
5	TIMING	NOT MET	43	11	18	37	2.966
6	**10**	**2.997**	**43**	**11**	**9**	**14**	**1.974**
7	50	3.924	41	11	8	18	1.789
8	100	7.848	41	11	8	18	1.732
9	5		*				
10	10		*				
11	50		*				
12	100		*				

Table 5.2: Post-Implementation results of the most interesting solutions. * marks the solutions whose implementation failed due to a lack of resources.

Of course, connecting the various blocks together is not enough to make

the MLP work on the FPGA. Indeed, the firmware of the PS needs to be written before to program the board, but this is out of the scope of this thesis. However, a short description of the behavior that the system should have is given anyway. At first, the PS triggers the PL by the AXI-Lite Interface. Subsequently, the PL reads data directly from the BRAM with its Input AXI-Master Interface, computes the output, and writes it back to the memory through the Output AXI-Master Interface. Finally the PL warns the PS that its execution is done with an interrupt, so that the latter can read the result from the BRAM. In this reasoning it is assumed that the data are already present in memory.

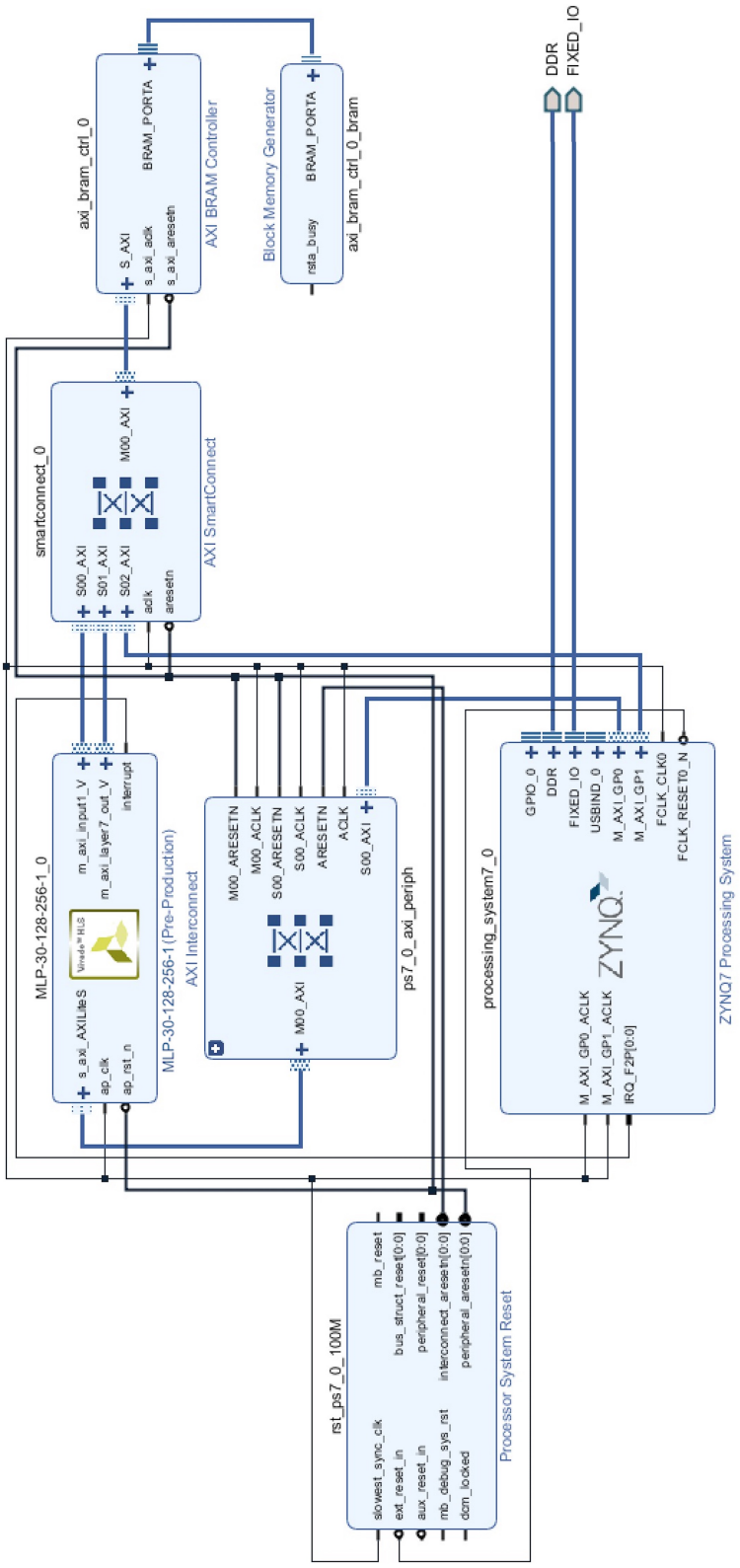


Figure 5.3: Block diagram of the HW implementation in the Vivado environment.

Chapter 6

Conclusion and Future Works

This thesis work showed that it is possible to detect contaminants accidentally included in packaged foods with Microwave Sensing combined with Machine Learning. The experiments carried out on cocoa-hazelnut spread jars are encouraging: the best Support Vector Machine reached a 10-folds CV accuracy of 95.052%, the best Multilayer Perceptron a 5-folds CV accuracy of 95.833%, both mispredicting 29 samples out of the 480 of the Real Test Set composed by only six types of foreign bodies. Moreover, the MLP implemented in FPGA is able to guarantee real-time detection thanks to its low latency of about 3 ms, a crucial element to sustain the throughput of a production line. Hence, the *MIT-Food prototype system* is on the good way to become an important additional block for the food industries that want to improve their intrusion detection rate of low density contaminants for food safety.

In the end, some possible hints and perspectives to improve the *MIT-Food prototype* require to:

- enlarge the static real dataset to include more samples of contaminants already considered and to add new types of intrusions, such as wood;
- train different types of classifiers to see if they perform better than those used in this work. For example a *Nearest Neighbors*, a *Decision Tree*, a *Convolutional Neural Network*, an SVM with *threshold adjustment* [43] or an *ensemble classifier* [20] [28] made either by different classifiers or by

the same classifier adopted for each antenna pair/channel (e.g. *antenna grouping* [37]), as proven by [43];

- validate the classifiers with dynamic measurements, that is with a moving conveyor belt. For this purpose, place a photocell before the antennas arch to begin the acquisition when the jar is in the starting position. In addition, knowing the speed of the conveyor belt and the average time of an acquisition, the relative position of the jar with respect to the antennas arch can be calculated. So it could be feasible to discover which is the optimal switching sequence of active antennas pairs during the transit of the jar below the arch to maximize the illumination of the target [21] and to let the system work in high-speed conveyor belts;
- increase the speed of the acquisition with a 6-channel VNA, able to acquire all channels in parallel. In this way, a single row of the S-matrix is collected in one shot and the latency of the system is reduced;
- use more durable electromechanical switches able to sustain the industrial production rate in order to replace them not too often;
- integrate the VNA controller in hardware to have a compact device;
- validate the behavior of the system on different homogeneous food products, such as honey, yogurt, baby food, and on non-homogeneous ones, like chocolate spreads with hazelnut grains, to have a wide range of possible application scenarios and validate the prototype even more.

Bibliography

[1] Foreign Body (2017). *Oxford Dictionary of English*, Oxford University Press.

[2] Wright D., Friedrichs R. (2017). Foreign Bodies - Techniques for Investigation and Identification. https://www.rssl.com/%7E/media/rssl/en/files/documents/white-paper/rssl-foreign-bodies.pdf?la=en. Accessed: 30/10/2019.

[3] Giordano A., Vipiana F., Casu M. R., Savorani F. (2018). Microwave Imaging Technology for Food Contamination Monitoring. *Politecnico di Torino, Turin, IT*.

[4] White V. (2016). Mars recalls chocolate products in 55 countries. https://www.newfoodmagazine.com/news/22851/mars-recalls-chocolate-products-in-55-countries/. Accessed: 30/10/2019.

[5] Dellapiana, B. (2019). NUTKAO quality and safety. *MIT-Food - Microwave Imaging Technology for Food Contamination Monitoring*, Politecnico di Torino, Turin, 10 October. Accessed: 01/11/2019.

[6] Institute of Food Science and Technology, Manning L. (2012). Food and Drink - Good Manufacturing Practice - A Guide to its responsible management. *Institute of Food Science & Technology*, Wiley-Blackwell, 280. https://books.google.it/books?id=KugWswD7ESQC&printsec=frontcover&redir_esc=y#v=onepage&q&f=false. Accessed: 30/10/2019.

[7] Sperber W. H., Stier R. F. (December 2009). Happy 50th Birthday to HACCP: Retrospective and Prospective. *FoodSafety magazine*, 42–46. https://www.foodsafetymagazine.com/magazine-archive1/december-2009january-2010/happy-50th-birthday-to-haccp-retrospective-and-prospective/. Accessed: 30/10/2019.

[8] SAVORANI, F. (2019). MIT-Food Microwave Imaging Technology for

Food Contamination Monitoring: project overview. *MIT-Food - Microwave Imaging Technology for Food Contamination Monitoring*, Politecnico di Torino, Turin, 10 October. Accessed: 01/11/2019.

[9] Wang, Kaiqiang, Da-Wen Sun, Hongbin Pu. (2017). Emerging Nondestructive Terahertz Spectroscopic Imaging Technique: Principle and Applications in the Agri-food Industry. *Trends in Food Science & Technology 67*, 93-105.

[10] Peco InspX (2019). Detecting plastics with x-ray inspection systems. `https://www.peco-inspx.com/blog/x-ray-detectable-plastics/`. Accessed: 02/11/2019.

[11] sAwad T.S., Moharram H.A., Shaltout O.E., Asker D., Youssef M.M. (2012). Applications of Ultrasound in Analysis, Processing and Quality Control of Food: A Review. *Food Research International 48.2*, 410-27.

[12] Gowen A. A., Tiwari B. K., Cullen P. J., Mcdonnell K., O'Donnell C. P. (2010). Applications of Thermal Imaging in Food Quality and Safety Assessment. *Trends in Food Science & Technology 21.4*, 190-200.

[13] Vipiana, F. (2019). MIT-Food Microwave Imaging Technology for Food Contamination Monitoring: project overview. *MIT-Food - Microwave Imaging Technology for Food Contamination Monitoring*, Politecnico di Torino, Turin, 10 October. Accessed: 01/11/2019.

[14] Tobon V. J., Rivero J., Scapaticci R., Farina L., Crocco L., Vipiana F. (2019). Monitoring of Food Contamination via Microwave Imaging, *2019 International Applied Computational Electromagnetics Society Symposium (ACES)*, 1-2.

[15] Zhanke Yan, Yibin Ying, Hongjian Zhang, Haiyan Yu (2006). Research progress of terahertz wave technology in food inspection. *Proc. SPIE 6373, Terahertz Physics, Devices, and Systems, 63730R.*

[16] Nielsen M. A. (2015). Neural Networks and Deep Learning. *Determination Press.* Accessed: 14/11/2019. `http://neuralnetworksanddeeplearning.com/index.html`

[17] Conceicao R. C., Hugo Medeiros M., O'Halloran, Rodriguez-Herrera D., Flores-Tapia D., Pistorius S. (2014). SVM-based Classification of Breast Tumour Phantoms Using a UWB Radar Prototype System. *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*, 1-4.

[18] Byrne D., O'Halloran M., Jones E., Glavin M. (2011). Support Vector Machine-Based Ultrawideband Breast Cancer Detection System. *Journal of Electromagnetic Waves and Applications 25.13*, 1807-816.

[19] Sacristán J., Oliveira B. L., Pistorius S. (2016). Classification of Electromagnetic signals obtained from microwave scattering over healthy and tumorous breast models. *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1-5.

[20] Duda R., Peter E., Stork. D. G. (2001). Pattern classification. *Wiley, New York, 2nd Edition.*

[21] Ricci M., Crocco L., Vipiana F. (2019). Microwave Imaging Device for In-Line Food Inspection. Submitted at: *European Conference on Antennas and Propagation (EuCAP) 2020.*

[22] Migliaccio C. (2019). Non Destructive Control of Fruit Quality using Microwaves. *MIT-Food - Microwave Imaging Technology for Food Contamination Monitoring*, Politecnico di Torino, Turin, 10 October. Accessed: 01/11/2019.

[23] Vipiana F., Casu M., Vacca M., Dassano G., Turvani G., Tobon J., Demichela M., Geobaldo F., Savorani F., Bosco F., Mollea C., (2019). MIT-Food, Microwave Imaging Technology for Food Contamination Monitoring. Politecnico di Torino, Turin. Accessed: 01/11/2019.

[24] Sarwar I., Turvani G., Casu M. R., Tobon J. A., Vipiana F., Scapaticci R., Crocco L. (2018). Low-Cost Low-Power Acceleration of a Microwave Imaging Algorithm for Brain Stroke Monitoring. *Journal of Low Power Electronics and Applications, 8(4)*, 43.

[25] Berteo M., Boccacci P. (1998). Introduction to inverse problems in imaging. *Bristol Philadelphia: Institute of Physics.*

[26] Cortes C., Vapnik V. (1995). Support-Vector Networks. *Mach. Learn. 20, 3*, 273-297.

[27] Bishop C. (2006). Pattern recognition and machine learning (Information Science and Statistics). *New York: Springer Science Business Media.*

[28] scikit-learn developers (2019). scikit-learn user guide. *Release 0.21.3.*

[29] Hsu CW., Chang CC., Lin CJ. (2016). A Practical Guide to Support Vector Classification. *National Taiwan University, Taipei 106, Taiwan.* Accessed: 13/11/2019.

[30] Rosenblatt, F. (1958). The perceptron: A theory of statistical separability in cognitive systems (Project PARA). *Washington: U.S. Dept. of Commerce, Office of Technical Services.*

[31] Kingma D., BaAdam J. (2015). A Method for Stochastic Optimization. *3rd International Conference for Learning Representations, San Diego.*

[32] Glorot X., Bengio Y. (2010). Understanding the difficulty of training

deep feedforward neural networks. *13th International Conference on Artificial Intelligence and Statistics (AISTATS), Chia Laguna Resort, Sardinia, Italy.*

[33] Google Developers. Machine Learning Crash Course. `https://developers.google.com/machine-learning/crash-course`. Accessed 04/12/2019.

[34] Smith L. (2002). A tutorial on Principal Components Analysis `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`. Accessed: 05/11/2019.

[35] Hastie T., Tibshirani R., Friedman J. H. (2009). The elements of statistical learning: Data mining, inference, and prediction. *Springer, New York, 2 edition.*

[36] Tenenbaum J. B., de Silva V., Langford J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science 290*, 2319–2323.

[37] Oliveira B., Jones E., Science Foundation Ireland (2018). Towards Improved Breast Cancer Diagnosis using Microwave Technology and Machine Learning. *National University of Ireland Galway.*

[38] Abdulaal M., Casson A., Gaydecki P. (2018). Performance of Nested vs. Non-Nested SVM Cross-Validation Methods in Visual BCI: Validation Study. *2018 26th European Signal Processing Conference (EUSIPCO)*, 1680-1684.

[39] Bergstra J., Yamins D., Cox D.D. (2013). Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. *Proceedings of the 12th Python in Science Conference (SciPy 2013).*

[40] Bergstra J., Bardenet R., Bengio Y., Kégl B. (2011). Algorithms for Hyper-Parameter Optimization. *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*, Dec 2011, Granada, Spain.

[41] Koehrsen W. (2018). A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning. *https://towardsdatascience.com.*

[42] Frazier P. I. (2018). A Tutorial on Bayesian Optimization. *arXiv e-prints.*

[43] Santorelli A., Yunpeng L., Porter E, Popovic M., Coates M. (2016). Microwave Breast Cancer Detection via Cost-sensitive Ensemble Classifiers: Phantom and Patient Investigation. *Biomedical Signal Processing and Control 31.C*, 366-76.

[44] Ricci M. (2019). MIT-Food prototype: Multi-antenna system. *MIT-Food - Microwave Imaging Technology for Food Contamination Monitoring*, Politecnico di Torino, Turin, 10 October. Accessed: 01/11/2019.

[45] Tobon V. J. (2019). MIT-Food prototype: synthetic aperture system. *MIT-Food - Microwave Imaging Technology for Food Contamination Monitoring*, Politecnico di Torino, Turin, 10 October. Accessed: 01/11/2019.

[46] François C. and others (2015). Keras

[47] Santorelli A., Yunpeng L., Porter E., Popovic M., Coates M. (2014). Investigation of Classification Algorithms for a Prototype Microwave Breast Cancer Monitor. *8th European Conference on Antennas and Propagation (EuCAP 2014)*, 320-24.

[48] Duarte J. et al. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *JINST 13 P07027*.

Index