Machine Learning Assessment of Anti-Spoofing Techniques for GNSS Receivers

Gabriel Marchand[a], Abdelmalek Toumi[a], Gonzalo Seco-Granados[b] and José A. López-Salcedo[b]

[a] *Lab-STICC UMR CNRS 6285 ENSTA Bretagne, Brest, France*
[b] *Universitat Autònoma de Barcelona (UAB), IEEC-CERES, Bellaterra, Spain*

Abstract

Global Navigation Satellite Systems (GNSS) are often the target of malicious attacks and interferences, mainly spoofing, thus posing a significant threat to both civilian and military equipment, and therefore necessitating effective detection and identification of such attacks. In this 'Work-in-Progress' paper, we propose the application of Machine Learning neural networks, a methodology proven highly effective in fields like cyberattack detection, to identify spoofing events across various scenarios. Our approach consists in computing non-time related metrics from a dataset of known spoofed signals, using the observables and signal-level measurements provided by a GNSS software receiver. The training is validated on both spoofed and clean scenarios to ensure a comprehensive approach. Furthermore, we provide a description of the feature's importance in the decision-making process of the model.

1. Introduction

To meet the needs of the United States armed forces, the first model of a satellite geolocation system, TRANSIT, was implemented in the early 1960s. Developed by the U.S. Navy, TRANSIT was quickly followed by more sophisticated models, particularly the Global Navigation Satellite System (GNSS), which was the first satellite-based geolocation system open to civilians.s. Since then, the Geolocation and Navigation Satellite System (GNSS) technology has made enormous progress and has become indispensable for a plethora of uses in daily life. It has become a major economic and political issue, hence the development of the Galileo and Beidou-3 systems by European and Chinese respectively. Its architecture and functioning make it an easy target for malicious attacks: in the recent years, successful attempts of spoofing and jamming by rogue states or academic figures have shifted the debate on the urgent need to detect attacks and mitigate the damage. Civil GPS, which are embarked in civil transportation facilities, mainly boats or cars, are particularly vulnerable to this kind of threats but occurrences of jacking of secured military utilities have also been reported, e.g. Iran allegedly stole a US Air Force drone using low-cost spoofing [1].

Spoofing involves transmitting counterfeit GNSS-like signals to generate a false position in the target receiver without disrupting GNSS operations, ultimately gaining control over the receiver. This technique entails mimicking a false signal that shares the same code phase, carrier frequency, and Doppler frequency shift as the authentic navigation satellite signal, thereby enabling interference and signal capture.

Many counter techniques have been developed and published in the recent years, the majority focusing on analyzing signal level characteristics in order to trigger alarms when abnormal behavior is detected. They cover mainly disruption detection, for instance an abrupt change in either amplitude, beat carrier phase, code phase, and primarily power monitoring (RPM). In that sense, signal quality monitoring (SQM) consists in detecting distortions in the correlation peak that results from the correlation of the received signal and the local replica. For instance, among these SQM techniques, the most common metrics are often the delta and ratio tests [2], even though a wide range of similar metrics have also been proposed in the literature. Moreover, authentication of the signal by verifying the origin and integrity of signals is used to identify and prevent unauthorized or counterfeit transmissions [1].

WIPHAL 2023: Work-in-Progress in Hardware and Software for Location Computation, June 06–08, 2023, Castellon, Spain
gabriel.marchand@ensta-bretagne.org (G. Marchand); abdelmalek.toumi@ensta-bretagne.fr (A. Toumi); gonzalo.seco@uab.cat (G. Seco-Granados); jose.salcedo@uab.cat (J.A. López-Salcedo)
0009-0007-9749-6514 (G. Marchand); 0000-0001-8415-0871 (A. Toumi); 0000-0003-2494-6872 (G. Seco-Granados); 0000-0002-5392-6005 (J.A. López-Salcedo)

In the past two decades, the integration of machine learning into practical, real-world applications has experienced a remarkable surge, including the improvement of GNSS navigation performance.[3] Despite this massive expansion to very various domains, GNSS spoofing detection through machine learning implementation is fairly restrained [4, 5, 6, 7, 8]. Many of these works involve detection techniques that require PVT observables, and thus they cannot be applied until the GNSS receiver has succeeded in obtaining a position fix, which may take several seconds in cold start.

In this paper, we provide a comparative analysis of many of the existing anti-spoofing techniques when they are tested in a common framework comprising the same GNSS software receiver and the same input datasets of spoofed signals. To do so, a layer of machine learning algorithms is implemented in order to gather all the outputs provided by the considered anti-spoofing techniques, and then to determine the relevance of such techniques on the model decision process that is followed by the machine learning algorithms. Machine learning's greatest strength lies in its ability to discover patterns, outliers and hidden relationships in vast and intricate amounts of data, and therefore this feature is expected to help in unveiling which are the most effective anti-spoofing techniques among those being considered. It is important to remark that the input data for the machine learning algorithms (in our case, the output of the anti-spoofing techniques) often requires pre-processing such data through techniques like removal, cross-combination, and scaling, rather than directly converting these data into binary data (i.e. hard-decisions) for triggering alerts. These pre-processed data is then used as input for the machine learning model, which generates binary output predicting the presence of spoofing [9].

Once the introduction and motivation of this paper has been introduced, the remaining of this paper is structured as follows. The problem statement and the specific tasks that are conducted in the present work are briefly presented in Section 2. Next, a high-level explanation of the mathematics and technology used in this work is introduced in Section 3. The fundamentals of the considered Machine Learning architecture and its explicit the features are discussed in Section 4. Finally, the experimental setup is described in Section 5 and conclusions are drawn in Section 6.

2. Problem Statement

This work has two primary objectives. Firstly, we aim at developing a robust and efficient spoofing detection system that leverages machine learning techniques, including both traditional algorithms and neural networks to practically predict a malicious spoofing attempt. In parallel, we collect metrics recently introduced in the literature along with exclusive metrics to retrospectively evaluate their weight in the machine learning process. The ultimate goal is to establish a ranking that reflects the effectiveness of each metric in detecting spoofing attempts. By combining a sophisticated detection system with a comprehensive understanding of the factors influencing its success, we hope to enhance the overall reliability and practicality of our approach.

To achieve this, we will:

- Analyze multiple datasets containing both genuine and spoofed GNSS signals.
- Investigate, extract and build relevant features and metrics for spoofing detection using a software receiver.
- Train and evaluate a machine learning neural network to identify the most effective approach.
- Assess the impact of different features on the decision-making process of the models.
- Validate the performance of the developed system in realistic scenarios.

3. Signal Model and Software Receiver

3.1. Signal Model

A typical GNSS signal can be represented in a simplified manner by Eq (1) [10],

$$s(t; A, f_c, \phi) = Ad(t)c(t)\cos(2\pi f_c t + \phi(t)) \tag{1}$$

where $s(t; A, f_c, \phi)$ is the received signal at time t, A is the amplitude of the signal, which is related to the signal power, $d(t)$ is the data-modulated signal containing the bits of the navigation message, $c(t)$

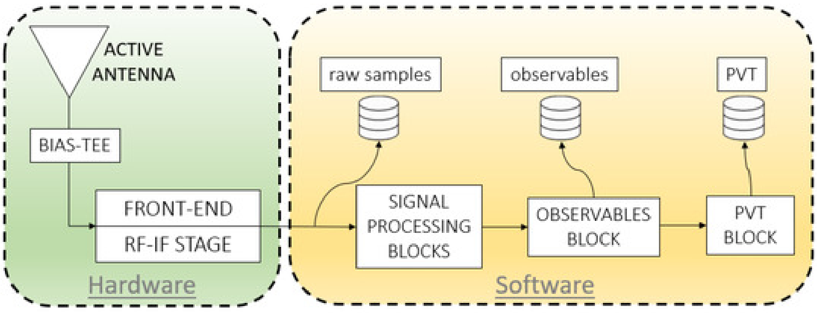

Figure 1: Software-Defined Receiver (SDR) architecture.

is the spreading code or the pseudorandom noise (PRN) code, f_c is the carrier frequency and $\phi(t)$ is the carrier phase.

In a spoofing attack, the attacker generates a counterfeit GNSS signal, $s_{\text{sp}}(t)$, that closely resembles the genuine signal, $s_{\text{gn}}(t)$. The goal is to deceive the GNSS receiver into locking onto the counterfeit signal instead of the genuine one. The received signal in the presence of a spoofing attack can be modeled by Eq (2) given in [5]:

$$r(t) = s_{\text{gn}}(t - \tau_{\text{gn}}; A_{\text{gn}}, f_{c,\text{gn}}, \phi_{\text{gn}}) + s_{\text{sp}}(t - \tau_{\text{sp}}; A_{\text{sp}}, f_{c,\text{sp}}, \phi_{\text{sp}}) + n(t) \tag{2}$$

where $s_{\text{gn}}(t)$ is the genuine GNSS signal and $s_{\text{sp}}(t)$ is the counterfeit GNSS signal generated by the spoofer, τ is the propagation delay and $n(t)$ is the additive noise.

The signal model forms the basis for feature extraction and analysis in the subsequent stages of our spoofing detection system. By exploring the differences between genuine and counterfeit GNSS signals, we can identify meaningful features and metrics that can be used as input for the machine learning models.

3.2. Software Receiver: A High-Level Overview

A GNSS software receiver is a GNSS receiver that processes the received signals using software algorithms. It consists of several modules working together to acquire, track, and decode the GNSS signals. These modules operate in a synchronized manner to compute the position, velocity, and time (PVT) information [11]. The main components of a software receiver are summarized in Fig. 1 and briefly described below:

- *Signal Acquisition:* This module searches for and acquires the GNSS signals transmitted by the satellites. It correlates the incoming signals with locally generated replicas of the PRN codes to determine the time delay and Doppler frequency shift.

- *Signal Tracking:* Once a signal is acquired, the tracking module continuously adjusts the local replicas of the PRN code and carrier frequency to keep them aligned with the incoming signal. This process involves adjusting the code delay and carrier frequency using tracking loops, such as the code phase tracking loop and the carrier phase tracking loop.

- *Demodulation and Decoding:* After tracking the signals, the receiver demodulates and decodes the navigation data. This information includes satellite ephemeris, clock corrections, and other auxiliary data. These data are essential for calculating the PVT solution.

- *PVT Calculation:* The receiver uses the decoded navigation data and the measured pseudoranges to compute the PVT solution. This calculation involves solving a set of nonlinear equations, which can be done using various algorithms, such as the least squares method or the extended Kalman filter.

4. Neural Network, Features and Input Data

4.1. Neural Network

Neural Networks are a subset of machine learning, mimicking the structure and function of biological neural networks, like the ones found in the human brain. It is typically structured with an input layer, one or multiple hidden layers and an output layer, they are well suited for modeling non-linear relationships. Therefore they have been increasingly used in various fields, from image recognition to language processing [5]. In this work, we choose to focus on one in particular:

The *Multi-Layer Perceptron* is a type of feedforward artificial neural network that consists of multiple layers of interconnected neurons. These connections have associated weights that are adjusted during the training process. The network learns to optimize these weights using an algorithm such as backpropagation, which minimizes the error between the predicted output and the actual target values. MLP is convenient in our case for multiple reasons. Firstly, it is a complex enough model to be able to handle complex and high-dimensional data to sense the outliers and patterns in non-linear relationships. Secondely, despite being highly efficient in most cases, it remains a fairly straightforward and computationally light algorithm in our particular architecture. This efficiency also facilitates the empirical search for optimal parameters through methods such as grid search[12].

4.2. Features Descriptors

To train and evaluate the performance of our machine learning models, we extract a set of features from the received GNSS signals. The following metrics are used for feature extraction[13] [14]:

- *Carrier to noise spectral density (C/N_0)*: It measures the strength of the GNSS signal relative to the noise level. It is an essential metric to assess signal quality and can be affected by spoofing attacks.

  The C/N_0 is computed by first calculating the signal power P and noise variance σ^2 from the tracked signal. Specifically, the signal power is calculated as $P = I_P^2 + Q_P^2$, where I_P and Q_P represent the in-phase and quadrature components of the signal. We plot it in Fig. 2. In that case, the scenario being a 10 dB power advantage, the metric is expected to raise, suggesting there might be an issue. The dataset involves a 2 ms (milliseconds) time delay or 'push', also described as a 'two chip delay', the signal is delayed by the duration of two pulses. In our case the software receiver, never being able to locate the newest strongest peak, is keeping track of the authentic signal instead of tracking the spoofed signal.

  The noise variance σ^2 is obtained from the deviation of the noise level from its mean over a certain interval. The C/N_0 is then given by the formula:

  $$C/N_0 = 10 \cdot \log_{10}\left(\frac{P}{\sigma^2} \cdot \frac{1}{PDI}\right) \tag{3}$$

  where the PDI factor is the time duration over which the signal power and noise variance are averaged.

- *The delta metric* ($\Delta\tau(t)$): is used as a metric for detecting spoofing and it is based on computing the following ratio [13]:

  $$\Delta\tau(t) = \frac{I_{E,\tau}(t) - I_{L,\tau}(t)}{2I_P(t)} \tag{4}$$

  In this equation, $I_{E,\tau}(t)$ and $I_{L,\tau}(t)$ correspond to early and late taps, respectively, which are τ seconds ahead and behind the prompt tap $I_P(t)$ in the in-phase component at time t. Since the delta test exhibits symmetry, $E[\Delta\tau(t)] = 0$ under conditions free from multipath and spoofing.

- *Quadrature discriminant metric*: This metric is similar to the delta metric, but for the quadrature component. It is represented as Q_{discr} and calculated as follows in Eq (4):

  $$\mathrm{Q}_{\mathrm{discr}} = \frac{Q_{L,\tau}(t) - Q_{E,\tau}(t)}{2Q_P(t)} \tag{5}$$

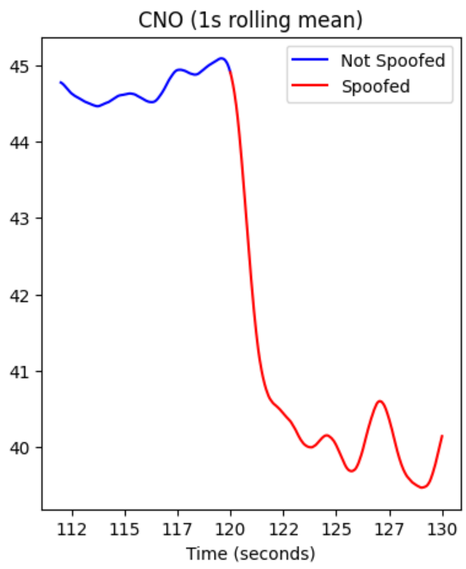

Figure 2: C/N_0 observables obtained ~ 10 seconds before and after the spoofer appeared in scenario 'os2'.

- *Early-late phase metric (ELP_τ)*: This metric captures the phase difference between early and late taps. It can help detect spoofing and multipath effects. It is calculated in Eq (5):

$$\mathrm{ELP}_\tau = \arctan\left(\frac{Q_{L,\tau}(t)}{I_{L,\tau}(t)} - \frac{Q_{E,\tau}(t)}{I_{E,\tau}(t)}\right) \tag{6}$$

- *In-phase and quadrature ratio discriminant metrics (I_{RD} and Q_{RD})*: These metrics measure the ratio between the sum of early and late taps and the prompt tap for both in-phase and quadrature components, respectively, as given in Eq (6) and Eq (7):

$$I_{\mathrm{RD}} = \frac{I_{L,\tau}(t) + I_{E,\tau}(t)}{2I_P(t)}, \tag{7}$$

$$Q_{\mathrm{RD}} = \frac{Q_{L,\tau}(t) + Q_{E,\tau}(t)}{2Q_P(t)} \tag{8}$$

- *Magnitude Difference Metric (MD_τ)*: This metric evaluates the difference between the magnitudes of the early and late taps normalized by the prompt tap magnitude. It can help detect spoofing attacks. It is calculated in Eq (8):

$$\mathrm{MD}_\tau = \frac{\sqrt{I_{E,\tau}(t)^2 + Q_{E,\tau}(t)^2} - \sqrt{I_{L,\tau}(t)^2 + Q_{L,\tau}(t)^2}}{\sqrt{I_P(t)^2 + Q_P(t)^2}} \tag{9}$$

- *Q-channel Signal Quality Monitoring (SQM) metric*: It is also denoted as M_{sqm}, is a novel SQM metric proposed in [15] to address the limitations of traditional SQM metrics, such as limited spoofing detection accuracy and low robustness due to false alarms caused by environmental effects like multipath. Traditional SQM metrics are mainly constructed based on the in-phase correlator outputs in the tracking loop of a GNSS instrument. During a spoofing attack, the interaction between genuine and fake signals may lead to a transfer of correlation energy into the quadrature channel. This abnormal quadrature channel energy serves as the primary indicator for the M_{sqm} metric. In the absence of spoofing, the typical value of Q-channel energy is 0, while the typical value of I-channel energy is large and nonzero. When spoofing is present, even a weak abnormal energy can quickly appear in the Q-channel. The metric is given in Eq (9):

$$M_{\mathrm{SQM}} = \sqrt{\left(\frac{Q_{E,\tau}(t)}{I_P(t)}\right)^2 + \left(\frac{Q_{L,\tau}(t)}{I_P(t)}\right)^2} \tag{10}$$

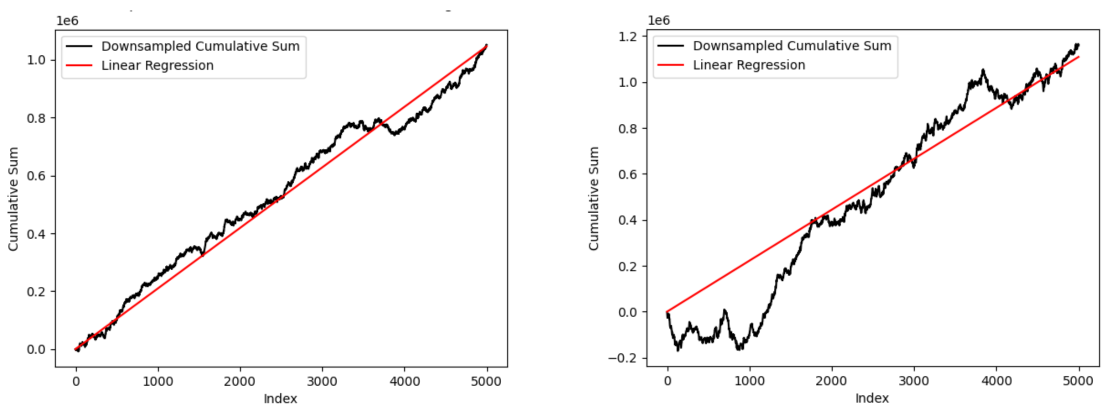

Figure 3: Cumulated sum and its linear regression for snapshot for scenario 'os2' in the absence (left) and presence (right) of spoofing.

- *Partial Correlation*: It is computed by means of the cumulative sum of the samples comprising a bit period. When attempting spoofing, the attacker will struggle to instantly know what to output in the first instant of a bit sign change, thus often giving a null or random output. This phenomenon is highlighted in the right hand of Fig. 3, where the presence of random guesses of the bit during the first samples of the bit period causes the cumulative sum to stagnate for a while. In contrast, for the case of a clean signal, the cumulative sum follows its linear regression nicely. The idea of this technique is to exploit this phenomenon by computing the correlation between the first and last samples of the bit period [16].

- *Custom CCAF Metric*: We compute the Cross-Ambiguity Function (CCAF) periodically, every second before the tracking loop when this computation is traditionally only done during the acquisition process. The CCAF can be useful to spot the presence of multiple signals even in cases where the spoofing is inaccurate and the counterfeit peak is significantly distant from the authentic signal by computing the correlation over both the Doppler frequency and code delay [17]. On the other hand, the computation is heavier and longer in time than other metrics proposed. In the matlab code, we reduced the calculation in an specific area around the first peak found: the computation is faster but the spoofer signal's peak could be missed. In our custom metric, we analyze the CCAF to identify the two largest peak values, which can correspond to the authentic and counterfeit signals: after finding the first peak, we erase and replace a given area around this peak by the average noise level before grid searching for the highest peak.
The metric is computed using with Eq (10):

$$\lambda = \frac{R_1 - R_2}{R_1} \tag{11}$$

where R_1 represents the largest peak value and R_2 represents the second-largest peak value of the CCAF. This custom metric helps us assess the impact of spoofing on the CCAF over time, by quantifying the relative difference between the two most prominent peaks. As an example, Fig. 4 displays the CCAF when there is spoofing, where one can see that two peaks do appear, thus indicating the presence of two simultaneous GNSS signals, one of which must be the spoofed signal.

- *SAM Metric*: We adapted a metric originally designed for multipath detection, as described in [18], to suit our needs. The Slope Asymmetry Metric (SAM) is based on comparing the left and right slopes of the received signal correlation peak. Ideally, both slopes should be equal (but sign reversed), and their sum should be close to zero. The metric was developed primarily for static multipath scenarios, but we have adjusted it to be applicable in our context. Using Least Squares Regression on four points of each side of the correlation function centered on its peak, the sum of the two slopes is the metric used to detect any spoofing. Starting from the least squares regression, we can use the normal equations to find the slopes for the left and right sides of the correlation function. Given 4 points on each side, we can express the problem in matrix form. Let $\mathbf{A}_L$ and $\mathbf{A}_R$ be the 4×2 matrices for the left and right sides respectively, where the first column contains the time shift values x and the second column is filled with ones:

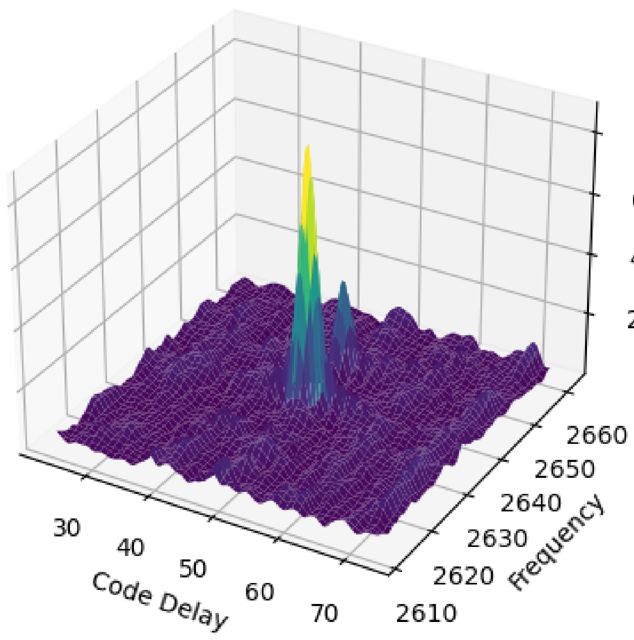

Figure 4: CCAF plot when spoofing, zoomed 20 digits around main peak.

$$\mathbf{A}_L = \begin{bmatrix} x_{1L} & 1 \\ x_{2L} & 1 \\ x_{3L} & 1 \\ x_{4L} & 1 \end{bmatrix}, \quad \mathbf{A}_R = \begin{bmatrix} x_{1R} & 1 \\ x_{2R} & 1 \\ x_{3R} & 1 \\ x_{4R} & 1 \end{bmatrix} \tag{12}$$

Let $\mathbf{y}_L$ and $\mathbf{y}_R$ be the 4×1 column vectors containing the corresponding correlation values:

$$\mathbf{y}_L = \begin{bmatrix} y_{1L} \\ y_{2L} \\ y_{3L} \\ y_{4L} \end{bmatrix}, \quad \mathbf{y}_R = \begin{bmatrix} y_{1R} \\ y_{2R} \\ y_{3R} \\ y_{4R} \end{bmatrix} \tag{13}$$

The normal equation for each side can be expressed as:

$$\mathbf{A}_L^T \mathbf{A}_L \begin{bmatrix} m_L \\ b_L \end{bmatrix} = \mathbf{A}_L^T \mathbf{y}_L, \quad \mathbf{A}_R^T \mathbf{A}_R \begin{bmatrix} m_R \\ b_R \end{bmatrix} = \mathbf{A}_R^T \mathbf{y}_R \tag{14}$$

These normal equations can be solved to obtain the slope and intercept (m_L, b_L, m_R, b_R) for each side. In matrix notation, these solutions would be:

$$\begin{bmatrix} m_L \\ b_L \end{bmatrix} = (\mathbf{A}_L^T \mathbf{A}_L)^{-1} \mathbf{A}_L^T \mathbf{y}_L, \quad \begin{bmatrix} m_R \\ b_R \end{bmatrix} = (\mathbf{A}_R^T \mathbf{A}_R)^{-1} \mathbf{A}_R^T \mathbf{y}_R \tag{15}$$

After obtaining the slopes m_L and m_R, we can calculate the Slope Asymmetry Metric (SAM) as the sum of the two slopes in Eq (15):

$$\text{SAM} = m_L + m_R \tag{16}$$

A significant deviation of the SAM metric from zero would indicate potential spoofing.

4.3. Input Datasets

In this study, we employ two publicly available datasets to evaluate the performance of our machine learning models, namely the Texas Spoofing Test Battery (TEXBAT) provided by the University of Texas at Austin, and the Oak Ridge Spoofing and Interference Test Battery (OAKBAT) from the US Oak Ridge National Laboratory. These datasets consist of diverse scenarios of spoofed GPS L1 C/A signals, as well as the baseline clean scenarios.

- *TEXBAT dataset* [19]: The Texas Spoofing Test Battery (TEXBAT) is a collection of six high-fidelity digital recordings of live static and dynamic GPS L1 C/A spoofing tests, conducted by the Radionavigation Laboratory of the University of Texas at Austin. The purpose of TEXBAT is to serve as the data component of an evolving standard aimed at defining spoof resistance for civil GPS receivers. The recordings capture a wide range of bandwidth and quantization considerations to support the evaluation of various authentication techniques. TEXBAT includes six spoofing attack scenarios and two clean datasets, with parameters such as spoofing type, platform mobility, power advantage, frequency lock, noise padding, and file size. The battery enables researchers to develop and evaluate spoofing detection techniques by studying the response of GPS L1 C/A receivers to the different spoofing attack scenarios presented in TEXBAT [20].

- *OAKBAT dataset [21]*: The OAKBAT dataset is a more recent dataset explicitly designed for GNSS spoofing detection research. It has been developed following the same methodology as the TEXBAT dataset to provide more data for researchers as those specific resources have proven to be scarce. This collection of digitized RF signals serves as both a complementary ”sibling” and an advancement to the widely used TEXBAT dataset. It comprises both authentic and spoofed GNSS signals collected in controlled environments, with the latter being generated using a commercially available signal simulator. The OAKBAT dataset consists of 16 unique datasets, with eight sets containing only the GPS L1 C/A signal and another eight sets containing only the Galileo E1 signal. Each group has two spoof-free, clean baseline sets and six sets with various degrees and types of spoofing. The datasets share several common parameters and are designed with reproducibility and accessibility in mind, making it an invaluable resource for researchers in the field of GNSS security and robustness.

5. Results

5.1. Experimental Setup

Our experimental setup comprises three main components: downloading spoofing or clean scenarios from the TEXBAT and OAKBAT datasets, processing the scenarios through a modified version of our software receiver to extract and build desired metrics, and building a machine learning neural network using these metrics as input.

1. Datasets: We aimed to include a diverse range of scenarios from both TEXBAT and OAKBAT datasets. In our experiments, we specifically used the Scenario 2 dataset from the TEXBAT collection, referred herein as 'ds2'. This scenario, also known as Static Overpowered Time Push (SOTP), features a spoofing attack where the spoofer has a 10 dB power advantage over the authentic signal ensemble. This scenario showcases the effects of a timing attack with a significant power advantage, forcing authentic signals into the noise floor and making the interaction between authentic and counterfeit signals less apparent. We also used the scenario 2 of the OAKBAT dataset, herein referred as 'os2', which gives similar features. To have a more complete training, we also used the two clean of spoofing static scenarios from both datasets. As a validation feature, scenario 4, herein referred as 'os4', of the OAKBAT dataset will be utilized.

2. Data Processing: We employed the FGI-GSRx software receiver for signal processing and analysis, which is provided as a companion software to [22]. Developed by the Finnish Geospatial Research Institute (FGI), this versatile GNSS software-defined radio tool is built with MATLAB, allowing users to process and analyze GNSS signals, which is particularly useful for research purposes. In our study, we used the FGI-GSRx software to evaluate the performance of spoofing detection techniques. We input the entire dataset into the software receiver but we focus on a shortened sample, which spans from approximately 10 seconds before the start of spoofing to 10 seconds after, according to the timestamps given by [19] for the TEXBAT dataset and [21] for OAKBAT.
For the non-spoofed scenarios, we compute from 100 to 120 seconds for the TEXBAT dataset and 110

to 130 seconds for the OAKBAT dataset. As a result, most metrics comprise 20,000 epochs, given that observations are obtained from the GPS L1 C/A signal every 1 ms. Some other metrics, such as those using the C/N_0, are obtained every 1 s. It is also important to remark that only one satellite is being processed at a time in the results to be shown next, in particular SV3 for TEXBAT and SV8 for OAKBAT.

3. Machine Learning: We implemented a feedforward neural network using python libraries TensorFlow and Keras to classify the presence of spoofing attacks based on the metrics extracted from the GNSS signals, with the goal of predicting the binary label. The choice to use a neural network was motivated by multiple reasons. The main reason is their ability to model complex relationships very well, which is not as efficient in different models like Support Vector Machines (SVMs) or the gradient boosting XGBoost. They are able to extract non-linear relationships and intricate interactions between features. Another reason is their flexibility in tuning the parameters and the architecture of the layer; as the training dataset is still to be updated with more various scenarios, the adaptability of Multi-Layer Perceptron (MLP) simplifies the modeling process. On the other hand, it is important to pay attention to overfitting and bias while training, some early setups really struggled to accurately predict the spoofing due to strong overfitting. To compare performances, we also implemented an SVM model in our preliminary stages. While the SVM model delivered results comparable to the MLP, we decided to concentrate on optimizing and detailing results from the neural network due to its aforementioned benefits.
We aim in this work at using binary classification to predict whether an epoch, here a 1 ms period, is spoofed or not. The algorithm consists of categorizing data into two classes: True or False. During the preprocessing, we first removed extreme outliers, mainly the first 1,000 points of the implementation, to eliminate any anomalous effects at the beginning of the dataset. Then, we applied a 1-second (1,000 points) rolling mean to smooth the data and reduce the noise. Thus, our input data is reduced from 20,000 epochs to 19,000 for 11 features. We also add the *spoofed* label that indicates whether a row is spoofed (equals 1) or not (equals 0).
The dataset is randomly divided between training and test datasets with a ratio 70%−30%. To this effect, we utilized the *train_test_split* function from the Scikit-learn Python library. This function first shuffles the dataset randomly, then allocates a specified proportion of data points to the training set and the remainder to the test set As a result, the training and test datasets are composed of 13 300 and 5700 points respectively.
Additionally, we computed the correlation between the different features to better understand their relationships and potentially reduce dimensionality as illustrated in Fig. 5: it displays the 11 features used before reducing the dimensionality as well as the label value *spoofed*. It is important to remark that the importance of the correlation is given by the absolute value: a value close to 1 or -1 means a strong relationship between the variable whereas the closer to 0 the weaker the link. Reducing dimensionality is primordial in making the training more efficient by getting rid of useless noisy features and particularly to avoid overfitting: other methods could be implemented such as Principal Component Analysis (PCA) to reduce the number of features using their variance [23].

To better training performance, the metrics of size 20,000 are scaled using MinMaxScaler following the formula:

$$X_{\text{std}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{17}$$

$$X_{\text{scaled}} = X_{\text{std}} * (\max - \min) + \min \tag{18}$$

where min, max are the features size.

In our model, we employed an L2 regularization technique to prevent any single feature from dominating the learning process. This technique adds a penalty proportional to the square of the magnitude of the weights to the loss function, discouraging the model from assigning too much importance to any particular feature. This helps in reducing overfitting and makes the model more generalizable [24].

The architecture of the implemented neural network is as follows:

- *Input layer*: A dense layer with 32 neurons and a ReLU activation function, which takes the feature vector with a length equal to the number of metrics.
- *Dropout layer*: A dropout layer with a dropout rate of 0.5 is added to prevent overfitting.
- *Hidden layer*: A dense layer with 16 neurons and a ReLU activation function. The Rectified Linear Unit layer (ReLU) is an activation function that simply retains positive inputs and sets

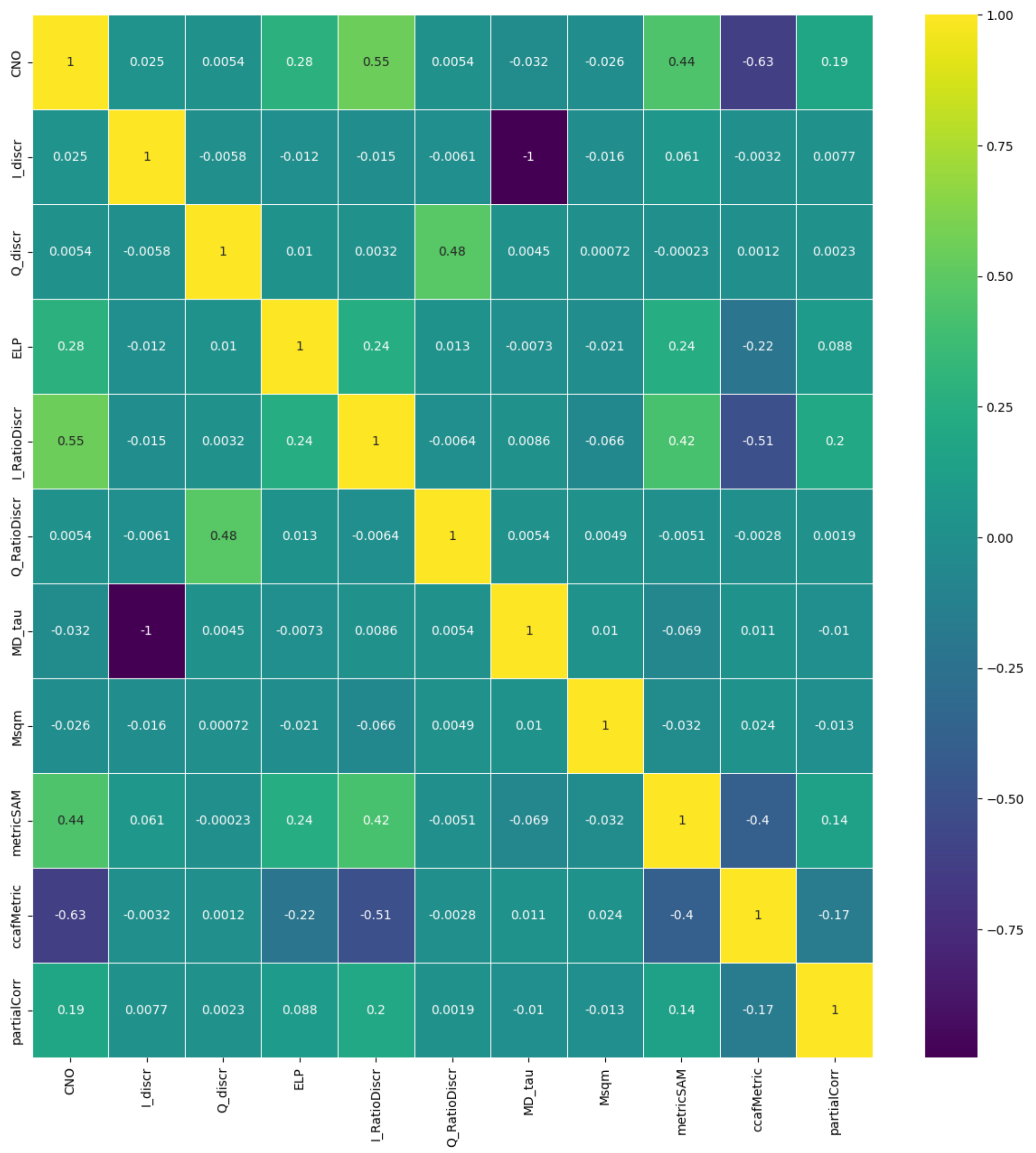

Figure 5: Correlation matrix in percentage for the TEXBAT dataset 'ds2'.

all negative inputs to zero. Although simple, this function has several interesting properties that make it very useful in neural networks. Firstly, while being a linear function for positive values, ReLU introduces non-linearity due to the threshold at zero, allowing neural networks with ReLU activations to model complex patterns and relationships. Secondly, ReLU leads to sparse activation, meaning that at any layer, some neurons can output a true zero, contrary to tanh and sigmoid functions that only can approach the zero value, making the network more efficient and easier to train. Thirdly, the computation is very straightforward and basic, making the computation very efficient, the function can be represented as $f(x) = \max(0, x)$[25].

- *Dropout layer*: Another dropout layer with a dropout rate of 0.5 is added.
- *Output layer*: A dense layer with a single neuron and a sigmoid activation function, as this is a binary classification problem.The sigmoid activation function, also known as the logistic function, is commonly used in the output layer of binary classification problems due to its ability to map any real-valued number into the range between 0 and 1. This makes it useful for outputting probabilities for the two classes in a binary classification problem. The sigmoid function is represented as $f(x) = 1/(1 + e^{-x})$ where x is the input to the function.

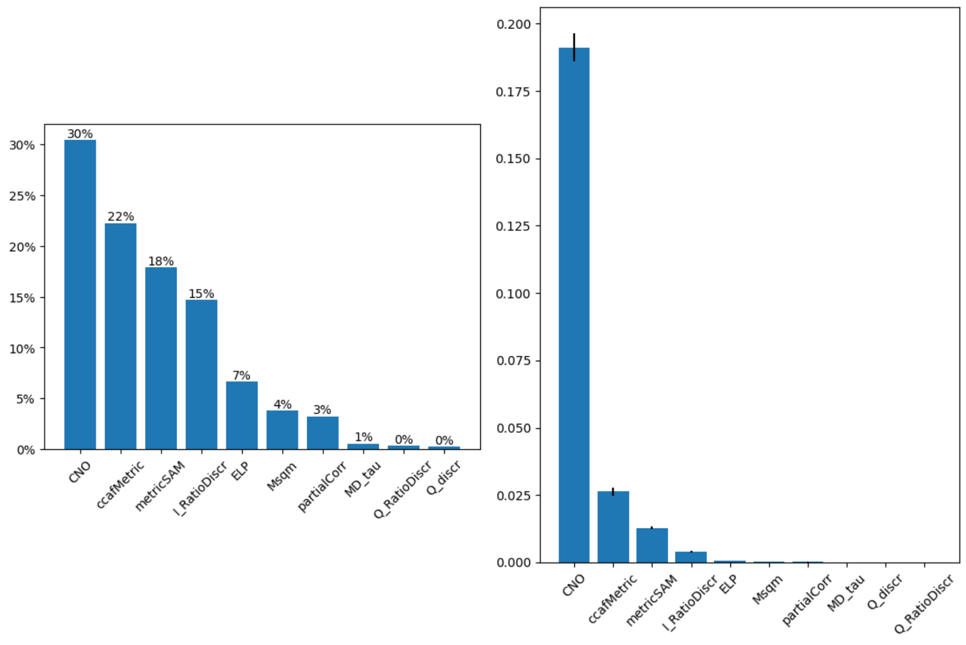

Figure 6: Bar plot of feature importance using connections weights (left) and permutation (right).

The model is compiled using the Adam optimization algorithm and the binary cross-entropy loss function, which is appropriate for a binary classification task. After hypertuning the parameter using a randomSearch, The model is trained on the training dataset for 30 epochs with a batch size of 32 and learning rate $\alpha = 0.01$. The performance of the model is evaluated on the test dataset using validation data during training.

5.2. Metrics Ranking

In this section, we present the results obtained from our experiments. The primary objective was to retrospectively identify the most influential metrics in our algorithm's decision-making process during training, in the idea to mitigate the "black-box" effect of such algorithms. At this point of time we build the ranking for a dataset composed of both TEXBAT and OAKBAT second scenarios and the OAKBAT clean static scenario. (ds2+os2+cleanStatic_os).
To compute the metrics importance, we use two different methods:

- *Feature importance using connections weights* This method aims at computing the relative importance of input features by calculating the weights of the neural network [26]. In our code, we evaluate the weight in between the input layer and the first hidden layer. The results are shown in the left hand side plot of Fig. 6.
- *Permutation importance* This method involves randomly shuffling the values of a single feature, running the model with the shuffled data, and measuring the change in performance. The larger the performance drop, the more important the feature. To visualize the results, we plot the importance of each feature with bar style, as well as the standard deviation error to estimate uncertainty in the right hand side plot of Fig. 6. It is worth noting that this does not represent a percentage because it doesn't involve breaking a whole into parts: it represents the average difference between the model with and without shuffling over multiple tries (here ten times for each feature). A feature could even have a negative value if shuffling it actually improves the model, for instance if the metrics only add noise to the model [27].

The first method uses directly the inner working of the neural network but treats the metrics independently, which could cause a wrong estimation of features' correlation and therefore output wrong results. The second method is not relying on the model's structure, thus can more adequately estimate more complex models despite being computionally heavier. We hope to have a better vision of the model computation by exhibiting those two methods together.

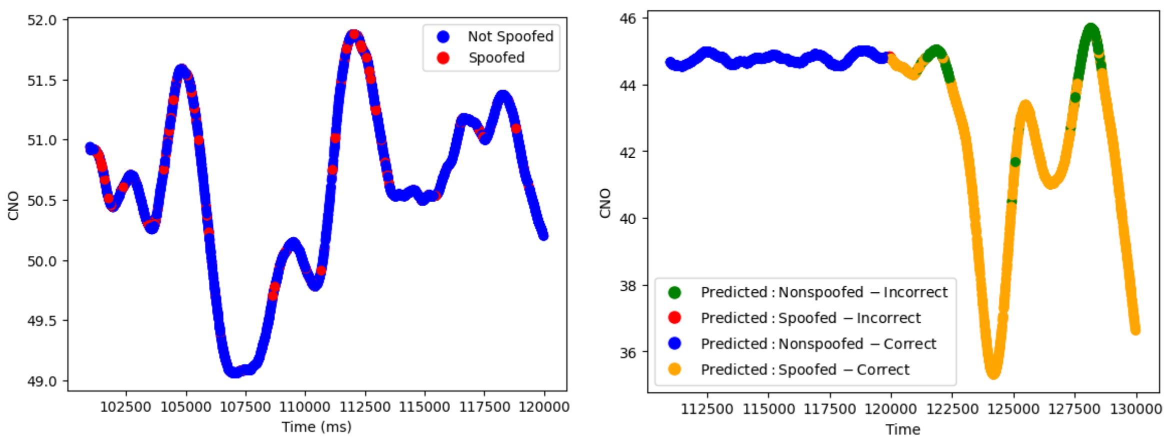

Figure 7: Epochs spoofing decisions using the TEXBAT 'cleanStatic' scenario (left) and the OAKBAT spoofed dataset 'os4' scenario (right).

Validation Dataset	Non-Spoofed period	Spoofed period
CleanStatic	81%	–%
os4	99%	79%

Table 1
Percentage of correctly validated epochs for each dataset.

	cleanStatic	os4
Top 5	99%	77%
Top 3	97%	75%

Table 2
Summary of results on both validation datasets when retaining the most predictive features.

5.3. Validation Results

In this section, we discuss the validation for our merged datasets (os2+ds2+oakbatCleanStatic). To ensure a comprehensive evaluation, we validate on two independent datasets: the spoofed OAKBAT scenario 'os4' and the clean TEXBAT 'cleanStatic' scenario. This approach allows us to gauge the algorithm's performance in detecting and differentiating between the two types of data.

Firstly, being sure that our algorithm does not wrongly overly identify spoofing events in a clean dataset is primordial: hereby the left hand side plot in Fig. 7 shows the C/N_0 over the acquisition stage of the GNSS receiver according to the predictions on the non-spoofed scenario. The results are quite promising: over 80 % of the epochs are predicted correctly.

In terms of the spoofed dataset, we plot the C/N_0 where the color-coded data points represent different prediction states in the right hand side plot of Fig. 7 alongside with the confusion matrix in Fig. 8: we observe that the model identifies almost perfectly the non-spoofed period (in blue, 98% precision) whilst it struggles more on the spoofed event (79%). This is confirmed by the confusion matrix Fig. 8: 417 spoofed epochs are being predicted wrongly.
For a practical use, we could convene that if the frequency of predicted spoofing exceeds a specified threshold (e.g., 75%) within a certain time frame, it is likely that a spoofing attempt is in progress. Tab. 1 summarizes all the validation results.
We proceeded to test our model using only the top five (CNO, ccafMetric, MetricSAM, ELP, Msqm) and top three (CNO, ccafMetric, MetricSAM, ELP, Msqm) metrics according to their importance given in Fig. 6. The model's performance degraded slightly: this suggests that even the lowest ranked metrics have valuable information, as illustrated in Tab. 2. Therefore, while our analysis confirms the significant impact of the top-ranked metrics, it also highlights the collective contribution of all metrics in achieving optimal detection accuracy. Regarding the clean Oakbat dataset, the results are sensibly the same.

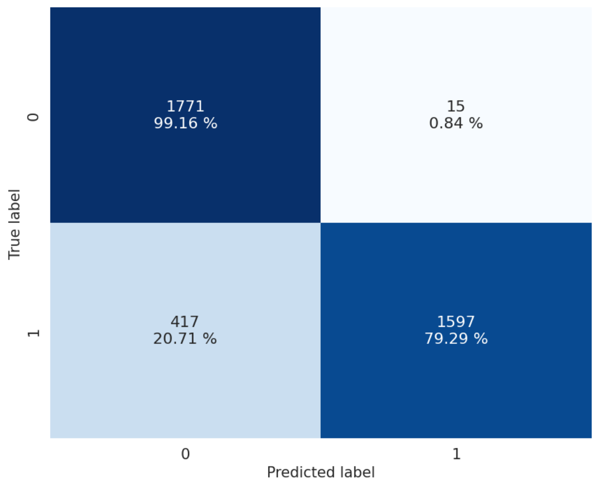

Figure 8: Confusion matrix for validation using the OAKBAT dataset 'os4'.

6. Conclusions

This work-in-progress article showcases an approach for developing an efficient machine learning-based spoofing detector concomitantly with a importance review of various metrics build in the process. This paper uses the neural network multi-layer perceptron neural network on selected metrics and combined datasets to classify artificially created spoofing scenarios. The procedure involves acquiring primary data through a software receiver, constructing the desired metrics mathematically, and applying classical machine learning pre-processing and scaling techniques.. The results give promising results as the algorithm is able to display spoofing and non spoofing events with correct results (with a minimum of 82% recall). As our range of validated datasets is fairly limited, we aim at improving the work by expanding the dataset with home generated spoofing scenarios using the Skydel GSG-8 GNSS simulator available at the SPCOMNAV research group. Those new datasets will enable a wider range of spoofing attacks, helping reduce the potential bias in our results.

Acknowledgments

This work was supported in part by the Spanish Agency of Research (AEI) under the Research and Development projects PID2020-118984GB-I00/AEI/10.13039/501100011033 and PDC2021-121362-I00/AEI/10.13039/501100011033.

Support for DOI 10.13139/ORNLNCCS/1664429 dataset (OAKBAT) is provided by the U.S. Department of Energy, project LOIS 9445 under Contract DE-AC05-00OR22725. Project LOIS 9445 used resources of the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] M. L. Psiaki, T. E. Humphreys, Gnss spoofing and detection, Proc. IEEE 104 (2016) 1258–1270. doi:10.1109/JPROC.2016.2526658.

[2] E. G. Manfredini, Signal processing techniques for GNSS anti-spoofing algorithms, Ph.D. thesis, 2017. doi:10.6092/POLITO/PORTO/2672749.

[3] A. Siemuri, H. Kuusniemi, M. Elmusrati, P. Välisuo, A. Shamsuzzoha, Machine learning utilization in GNSS—use cases, challenges and future applications (2021) 1–6. doi:10.1109/ICL-GNSS51451.2021.9452295.

[4] R. Morales Ferre, A. de la Fuente, E. S. Lohan, Jammer classification in GNSS bands via machine learning algorithms, Sensors 19 (2019) 4841. doi:10.3390/s19224841.

[5] P. Borhani-Darian, H. Li, P. Wu, P. Closas, Deep neural network approach to detect GNSS spoofing attacks (2020) 3241–3252. doi:10.33012/2020.17537.

[6] J. Winkel, Modeling and simulating gnss signal structures and receivers (2003).

[7] S. Semanjski, I. Semanjski, W. De Wilde, A. Muls, Use of supervised machine learning for GNSS signal spoofing detection with validation on real-world meaconing and spoofing data—part i, Sensors 20 (2020) 1171. doi:10.3390/s200411.

[8] S. Semanjski, I. Semanjski, W. D. Wilde, S. Gautama, GNSS spoofing detection by supervised machine learning with validation on real-world meaconing and spoofing data—part ii, Sensors 20 (2020) 1806. doi:10.3390/s20071806.

[9] P. Domingos, A few useful things to know about machine learning, Communications of the ACM 55 (2012) 78–87. doi:10.1145/2347736.2347755.

[10] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, S. H. Jensen, A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach, 2007. doi:10.1007/978-0-8176-4540-3.

[11] M. Cutugno, U. Robustelli, G. Pugliano, Low-cost GNSS software receiver performance assessment, volume 10, 2020, p. 79. doi:10.3390/geosciences10020079.

[12] F. Murtagh, Multilayer perceptrons for classification and regression, Neurocomputing 2 (1991) 183–197. URL: https://www.sciencedirect.com/science/article/pii/0925231291900235. doi:https://doi.org/10.1016/0925-2312(91)90023-5.

[13] E. Shafiee, M. Mosavi, M. Moazedi, Detection of spoofing attack using machine learning based on multi-layer neural network in single-frequency GPS receivers, Journal of Navigation 71 (2017) 1–20. doi:10.1017/S0373463317000558.

[14] K. Wesson, D. Shepard, J. Bhatti, T. Humphreys, An evaluation of the vestigial signal defense for civil gps anti-spoofing 4 (2011).

[15] C. Sun, J. W. Cheong, A. G. Dempster, H. Zhao, L. Bai, W. Feng, Robust spoofing detection for GNSS instrumentation using Q-channel signal quality monitoring metric, volume 70, IEEE, 2021, pp. 1–15. doi:10.1109/TIM.2021.3102753.

[16] G. Seco-Granados, D. Gómez-Casco, J. A. López-Salcedo, I. Fernandez-Hernandez, Detection of replay attacks to GNSS based on partial correlations and authentication data unpredictability, GPS Solutions 25 (2021). doi:10.1007/s10291-020-01049-z.

[17] S. Ahmed, S. Khanafseh, B. Pervan, GNSS spoofing detection based on decomposition of the complex cross ambiguity function, Navigation: Journal of The Institute of Navigation 68 (2021) 3569–3580. doi:10.33012/2021.17987.

[18] J. A. López-Salcedo, J. Parro-Jimenez, G. Seco-Granados, Multipath detection metrics and attenuation analysis using a GPS snapshot receiver in harsh environments, 2009, pp. 3692–3696.

[19] A. Lemmenes, P. Corbell, S. Gunawardena, Detailed analysis of the TEXBAT datasets using a high fidelity software GPS receiver (2016) 3027–3032. doi:10.33012/2016.14668.

[20] T. Humphreys, J. Bhatti, D. Shepard, K. Wesson, The Texas spoofing test battery: Toward a standard for evaluating GPS signal authentication techniques (2012).

[21] A. Albright, S. Powers, J. Bonior, F. Combs, A tool for furthering GNSS security research: The oak ridge spoofing and interference test battery (OAKBAT) (2020) 3697–3712. doi:10.33012/2020.17712.

[22] K. Borre, I. Fernández-Hernández, J. A. López-Salcedo, M. Z. H. Bhuiyan (Eds.), GNSS Software Receivers, Cambridge University Press, Cambridge, 2022. doi:10.1017/9781108934176.

[23] L. van der Maaten, E. Postma, H. Herik, Dimensionality reduction: A comparative review, Journal of Machine Learning Research - JMLR 10 (2007).

[24] A. Y. Ng, Feature selection, l1 vs. l2 regularization, and rotational invariance, in: Proceedings of the Twenty-First International Conference on Machine Learning, ICML '04, Association for Computing Machinery, New York, NY, USA, 2004, p. 78. URL: https://doi.org/10.1145/1015330.1015435. doi:10.1145/1015330.1015435.

[25] A. F. Agarap, Deep learning using rectified linear units (relu), arXiv preprint arXiv:1803.08375 (2018).

[26] J. D. Olden, An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data, Ecological Modelling 178 (2004) 389–397. doi:10.1016/S0304-3800(04)00156-5.

[27] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32. doi:10.1023/A:1010950718922.