ORIGINAL RESEARCH

Using Monte Carlo method to estimate the behavior of neural training between balanced and unbalanced data in classification of patterns

Paulo Marcelo Tasinaffo*, Gildárcio Sousa Gonçalves, Adilson Marques da Cunha, Luiz Alberto Vieira Dias

Technological institute of Aeronautics, Sao Jose Dos Campos/SP, Brazil

Received: March 26, 2018 **Accepted:** June 18, 2018 **Online Published:** July 31, 2018
DOI: 10.5430/air.v7n2p1 **URL:** https://doi.org/10.5430/air.v7n2p1

ABSTRACT

This paper proposes to develop a model-based Monte Carlo method for computationally determining the best mean squared error of training for an artificial neural network with feedforward architecture. It is applied for a particular non-linear classification problem of input/output patterns in a computational environment with abundant data. The Monte Carlo method allows computationally checking that balanced data are much better than non-balanced ones for an artificial neural network to learn by means of supervised learning. The major contribution of this investigation is that, the proposed model can be tested by analogy, considering also the fraud detection problem in credit cards, where the amount of training patterns used are high.

Key Words: Balanced data, Non-balanced data, Supervised learning, Artificial neural network, Monte Carlo method

1. INTRODUCTION

There are several books and papers mathematically describing the Monte Carlo method.[1–5] Nevertheless, it is needed a minimum knowledge of Statistics, in order to better understand this method. Classic bibliographic references on statistics and stochastic processes can be found in.[6–8] Monte Carlo method is a way of solving problems by using random numbers. This method is largely used for computational simulation models. In this context, simulation is defined as a technique that emulates an operation of real world system insofar, as this system evolves in time. An appropriate way of simulating the behavior of variable types to be analyzed is through the development of a simulation model by using probability distributions of discrete event known as Monte Carlo method.

According to Metropolis,[1] it is possible to deliver complex analytical mathematic models, which modeled the iterations between nuclear particles, in order to model the rules and statistics that governs each stage of a process. Starting from a uniform statistical distribution and mapping it, for the distributions interesting to a given problem, it was possible to process multiple decisions chains (simulations) and draw out relevant outcomes with an accuracy that is compatible to the one relevant for analytical methods.

The use of artificial intelligence has allowed great advances in the computational area.[11, 16, 18] The most successful approach has been the use of the artificial neural network, allowing, for example, writing recognition, image processing, modeling of nonlinear dynamic systems, applications in control theory, among others. A good introduction to artificial

*Correspondence: Paulo Marcelo Tasinaffo; Email: tasinaffo@ita.br; Address: Technological institute of Aeronautics, Sao Jose Dos Campos/SP, Brazil.

neural networks theory can be found in.[11, 18, 19] A very important starting point in the study of artificial neural networks is that they are considered universal approximators of functions.[9, 10, 12–15, 17]

In study[20] the problems of classifying patterns involving unequal probabilities in each class are treated and discussed in detail. In it, the authors discuss metrics for systems that use neural networks of multilayer perceptrons (MLP) for the task of classifying new patterns. In study[21] a new semi-supervised probabilistic detection structure is proposed using a special neural network structure called Probability Posterior Support Vector Machines (PPSVM) to classify unbalanced data. So the hierarchical probabilistic model of PPSVM is used for anomalies detection in order to alleviate unbalanced data problems with small samples and to improve the accuracy of detection of unbalanced classes.

In study[22] the novel algorithm based on analysis of variance (ANOVA) combined with fuzzy c-means (FCM) and bacterial foraging optimization (BFO) are proposed to classify unbalanced data. The algorithm is supported by simulation results. In the results of this simulation, the accuracy of the classification of the proposed algorithm can surpass other existing approaches. In study[23] it is mentioned that the performance pattern of a neural network can be influenced by several factors. One of these factors is related to the considerable difference that may exist between the number of examples belonging to each class to be recognized. The so-called unbalanced data effect may negatively influence the ability of a neural network to learn the concept of the minority class.

Initially, the difference between balanced and non-balanced data should be explained. Non-balanced data are data where the amount of data with zero outputs and data with one output are discrepant, for example, 99% of patterns with outputs equal to zero and just 1% of patterns with outputs equal to one is a good example of non-balanced data. In a perfect case, balanced data occur when there are 50% of patterns with outputs equal to zero and 50% of patterns with outputs equal to one.

This paper intends to accomplish a Monte Carlo simulation to estimate the best mean square error to train an artificial neural network with feedforward architecture, for example, Multilayer Perceptron (MLP) networks or Radial Basis Functions (RBF). Two case studies will be considered involving balanced and non-balanced data networks. On these case studies, only the specific case of non-linear classification problem of patterns where the net outputs are zero or one will be considered. Next we mathematically define the problem that will be addressed throughout this article.

Definition of the problem addressed: To the training patterns of input of the neural network given by the vectors $\vec{x}^j$ for $j = 1, 2, \cdots, p$ and for $\vec{x}^j = [x_1^j . x_2^j, \cdots, x_n^j]^T$ where n is the input dimension of the neural network and for the output patterns of the neural network given by scalars y^j for $j = 1, 2, \cdots, p$, where p is the total number of training patterns, then the vectors $\vec{x}^j s$ and the scalars $y^j s$ make up the input/output patterns of the neural network. For the particular case of a pattern classification problem, the scalars $y^j s$ can assume only the discrete values $\{0, 1\}$. Let, by definition, be the scalar $p = n_1 + n_2$, where n_1 is the total number of training patterns that assume the value 1 and n_2 the total number of training standards that assume the value 0. Then, the scalar $n_{21} = \frac{n_2}{n_1}$ is defined as the balancing factor of the training patterns. The problem to be addressed here is to make some a priori estimates of the mean quadratic error of neural network training and to find out how they vary as a function of the balancing factor n_{21} for the particular problem of pattern classification. Because it is an estimate, it will not be necessary to perform any real neural training to obtain these values. It is intended to conclude mathematically, using the Monte Carlo Method, that unbalanced data require a more rigid mean square error than balanced data. Balanced data are those data where the value of n_{21} are very close to one and unbalanced data have values of n_{21} very far from the unit value. Only by mere convention to be followed from beginning to end of this article, the values associated with n_2 will express the sense of abundance of data and the values associated with n_1 will express the sense of data scarcity. There are several papers in the literature (see[19–23]) describing class imbalance in pattern classification problems. However, as described here, nothing was found about it.

This paper is divided into the following sections: In section 2, there is a brief description of the Monte Carlo method; section 3 presents a detailed analytical description of the behavior of the mean squared error of neural training for balanced and non-balanced data; section 4 presents a detailed flowchart of the proposed Monte Carlo algorithm; section 5 discusses some numerical outcomes obtained; and section 6 shows the conclusions, concerning the proposed method.

2. THE MONTE CARLO METHOD

The Monte Carlo method represents a way of solving problems by using random numbers. This method is largely used in computational simulation models. In this context, simulation is defined as a technique that emulates an operation of real world system, insofar as this system evolves in time. A suitable way of simulating the behavior of the type of variables that one intends to analyze, as it can be seen in Figure 1, is through the development of a simulation model,

by using discrete event probability distributions known as Monte Carlo method. By using the Monte Carlo method, it is possible to establish accurate geometrical considerations of the length and area of complex Figures from purely probabilistic considerations. This is due to its great ability of generating random numbers inherent to computers.

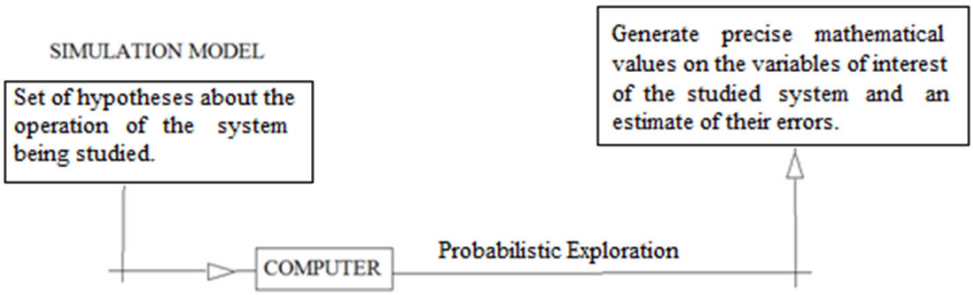

Figure 1. The use of computers in real world problems simulation through the Monte Carlo method

Let us explain Figure 1 through a simple example. Consider the problem of estimating the integral of function f over a unit interval. We can represent the integral

$$\alpha = \int_0^1 f(x)dx \tag{1}$$

as an expectation $E[f(U)]$, with U uniformly distributed between 0 and 1. Suppose we have a mechanism to generate random point $U_1, U_2, \cdots, U_n$ independently and according to a uniform distribution in the interval [0,1]. In this way, evaluating the f in these n points and calculating their mean, we obtain the following Monte Carlo estimate[4] for the integral (1):

$$\hat{\alpha}_n = \frac{1}{n}\sum_{i=1}^{n} f(U_i) \tag{2}$$

If f is actually integrable over the interval [0,1] then by the law of large numbers, $\hat{\alpha}_n \rightarrow \alpha$ with probability 1, when $n \rightarrow \infty$. However, the most interesting fact of calculating this integral is that the error of estimate $\hat{\alpha}_n$ can be determined by applying the standard deviation formula:[4]

$$e_\alpha \cong \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(f(U_i)-\hat{\alpha}_n\right)^2} \tag{3}$$

Thus, from the random values $f(U_1), f(U_2), \cdots, f(U_n)$ of the function f, we obtain not only an estimate of the integral (1), but also the error of this estimate.

An important consideration concerning the Monte Carlo method is that it is based on the real analogy between probability and volume, including the volume considered in n-dimensional space. The mathematics of measurement[4] formalizes the intuitive notion of probabilities by associating an event with a set of possible occurrences and defines the probability of an event to occur over the whole universe of occurrences. The Monte Carlo method uses this identity in an inverted way,[4] that is, by providing the volume calculation of a set through interpreting the volume as a probability. This means that random or aleatory samples of a universe of occurrences of interest — exhaustively generated by the computer — lead to a subset of laws that form that considered volume.

The Generation of Random Numbers - any computer simulation of a physical system that involves randomness includes a method to generate sequences of random numbers, for example, a simulation of line systems involves the generation of interval between the arriving of people and the time of attendance of each client. Random numbers should always fulfill the physical process properties they are simulating and the computer numerical simulation involves the generation of long sequences of random numbers, and for this reason, it involves the generation of random variables with pre-defined distributions.

Many numerical problems in science, engineering, finance, and statistics are currently solved by the Monte Carlo method, that is, by the means of random experiments performed in a computer. In such a way, the heart of any Monte Carlo method is a random numbers generator: a procedure that produces an infinite sequence

$$U_1, U_2, U_3, \ldots \overset{iid}{\sim} Dist \tag{4}$$

of random variables that are independent and identically distributed (iid), according to any probability distribution. The concept of an infinite iid sequence of random variables is a mathematical abstraction that might be impossible of being implemented in a computer. The maximum that a computer

can obtain is an approximation of that, because in practice, an infinite sequence of random numbers is impossible.

A vast common category of random numbers generators is based on simple algorithms that can easily be implemented in a computer. Those algorithms can usually be represented[5] for a tuple (S, f, μ, U, g) where:

- S is a finite set of states;
- f is a function from S to S;
- μ is a probability distribution over S;
- U is an output space, for example, for a uniform random generator, U is the interval (0,1);
- g is a function from S to U.

So, computationally, a random number generator should have the following structure:[5]

The Algorithm 3.1 (Generic random numbers generator):

(1) Starting - generate the seed S_0 of a probability distribution μ over S. Establish $t = 1$;
(2) Transition - Establish $S_t = f(S_{t-1})$;
(3) Output - Establish $U_t = g(S_t)$; and
(4) Repetition - Establish $t = t + 1$, and come back to step 2.

The above algorithm produces a sequence $U_1, U_2, U_3, \cdots$ of pseudo-random numbers. They can be referenced as random numbers only. A very important definition is the one of length period of a random generator numbers. Length period is, by definition, the fewest number of steps that an algorithm takes before entering again in an already visited state.

The most common methods to generate pseudo-random sequences use linear recurrent relations. Three types of random generators, much used in the literature, are:[5] *Linear Congruential Generators, Multiple-Recursive Generators, and Matrix Congruential Generators.*

Another important question is how to generate a sequence of random numbers, according to a normalized Gaussian distribution (mean $x_m = 0$ and standard deviation $\sigma_x = 1$) from a uniform distribution. This can be easily answered from the Central Limit Theorem that states that an addition of any probability distribution tends to be a Gaussian one, when the number of samples tends to infinity. It is important to note that all Monte Carlo simulations that will be performed in this article will consider only Gaussian distributions. The Laplace-Gauss probability distribution function is mathematically defined as:

$$G(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2} \tag{5}$$

where μ and σ are two constants or parameters and y is the continuous variable. It can be mathematically demonstrated that μ and σ are respectively the mean value and the standard deviation of this probability distribution.

It is interesting to notice that when n measurements of a random process indicated by the sets of measurements $\{y_1, y_2, \cdots, y_n\}$ are known and if there is evidence to be a Gaussian distribution, then mean and standard deviation values can be roughly calculated by equations (6.a) and (6.b) and these values can be substituted in equation (5) to determine a continuous Gaussian probability distribution for the n discrete values known.

$$\mu = \frac{\sum_{i=1}^{n} y_i}{n} \quad \text{(a)}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \mu)^2}{n-1}} \quad \text{(b)} \tag{6}$$

It is evident that the sample of data should be meaningful, in order that the continuous approximation may suitably adapt itself to the discrete values given.

3. THE ANALYTICAL ANALYSIS OF MEAN SQUARE ERRORS

In order to begin the analytical analysis of the mean square error, one will start from the mathematical definition of mean square error. This definition is expressed in the following equation (7)

$$e_m^2 = \frac{1}{n} \cdot \sum_{i=1}^{n} (\bar{y}_i - \hat{y}_i)^2 \tag{7}$$

where $\bar{y_i}$ is the desired value of the estimator and $\hat{y_i}$ is the value estimated by the estimator.

It is interesting to notice that for binary classification of pattern problems, for example, in determination of credit card frauds, where one has the value 1 to represent the fraud and the value 0 to represent the absence of frauds, one can divide the mean square error into two portions as indicated in equations (8.a) and (8.b)

$$e_m^2 = \frac{1}{n} \cdot \sum_{i=1}^{n_1+n_2} (\bar{y}_i - \hat{y}_i)^2 \quad \text{(a)}$$

$$e_m^2 = \frac{1}{n_1+n_2} \cdot [\underbrace{\sum_{i=1}^{n_1} (\bar{y}_i - \hat{y}_i)^2}_{=n_1 \cdot e_{m_1}^2} + \underbrace{\sum_{i=1}^{n_2} (\bar{y}_i - \hat{y}_i)^2}_{=n_2 \cdot e_{m_2}^2}] \quad \text{(b)} \tag{8}$$

where n is the total number of training patterns; n_1 is the number of training patterns with output 1 (*frauds or lacking data*); n_2 is the number of training patterns with output 0 (*absence of frauds or data in excess*); and $n = n_1 + n_2$. Being so, it is stipulated from now on that $e^2_{m_1}$ is the mean square error of the lacking data and $e^2_{m_2}$ is the mean square error of data in excess. The equation (8.a) can be simplified as follows:

$$\begin{aligned} e^2_m &= \frac{n_1}{n_1+n_2}\cdot e^2_{m_1} + \frac{n_2}{n_1+n_2}\cdot e^2_{m_2} \quad \text{(a)} \\ n\cdot e^2_m &= n_1\cdot e^2_{m_1} + n_2\cdot e^2_{m_2} \quad \text{(b)} \\ n_2\cdot e^2_{m_2} &= n\cdot e^2_m - n_1\cdot e^2_{m_1} \quad \text{(c)} \\ e^2_{m_2} &= \left(\frac{n}{n_2}\right)\cdot e^2_m - \left(\frac{n_1}{n_2}\right)\cdot e^2_{m_1} \quad \text{(d)} \end{aligned} \tag{9}$$

The equation (9.d) is the equation of a line in the form $y = q - px$ (notice that this line has a negative bending). In a first analysis on the consistency of equation (9.d), one should notice that when $n_1 = 0$, there is the specific case of $e^2_{m_2} = e^2_m$, that is, the total absence or lacking of data implies that $e^2_{m_2}$ only depends on e^2_m. In order to check this, it is enough to do the following calculation:

$$e^2_{m_2} = \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_{m_1} \tag{10}$$

being $n_1 = 0$ in equation (10), one has

$$\begin{aligned} e^2_{m_2} &= \frac{0+n_2}{n_2}\cdot e^2_m - \frac{0}{n_2}\cdot e^2_{m_1} \quad \text{(a)} \\ e^2_{m_2} &= e^2_m \quad \text{(b)} \end{aligned} \tag{11}$$

Something that should be noticed in relation to the line given equation (9.d) are the two points of the line that cut the coordinated axes. This significance is justified further up. This way, in order to calculate the point that cuts the abscissa, that is, the point $(0, e^2_{m_2})$, one does the following mathematical operation:

$$\begin{aligned} e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_{m_1} \quad \text{(a)} \\ e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot 0 \quad \text{(b)} \\ e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m \quad \text{(c)} \end{aligned} \tag{12}$$

The equation (12.c) implies that the point $(0, e^2_{m_2}) = (0, \frac{n_1+n_2}{n_2}\cdot e^2_m)$ belongs to the line (12.a). In order to calculate the point that cuts the coordinate axis, that is, the point $(e^2_{m_1}, 0)$, one does the following mathematical operations:

$$\begin{aligned} e^2_{m_2} = 0 &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_{m_1} \quad \text{(a)} \\ \frac{n_1}{n_2}\cdot e^2_{m_1} &= \frac{n_1+n_2}{n_2}\cdot e^2_m \quad \text{(b)} \\ e^2_{m_1} &= \frac{n_1+n_2}{n_1}\cdot e^2_m \quad \text{(c)} \end{aligned} \tag{13}$$

The equation (13.c) implies that the point $(e^2_{m_1}, 0) = (\frac{n_1+n_2}{n_1}\cdot e^2_m, 0)$ belongs to the line (12.a). From equations (12.c) and (13.c), one has the two pursued points, given by:

$$\begin{aligned} (0, e^2_{m_2}) &= (0, \frac{n_1+n_2}{n_2}\cdot e^2_m) \quad \text{(a)} \\ (e^2_{m_1}, 0) &= (\frac{n_1+n_2}{n_1}\cdot e^2_m, 0) \quad \text{(b)} \end{aligned} \tag{14}$$

Figure 2 illustrates the equation of the line given by equation (10). An analysis of the graphic of Figure 2 allows one to elaborate isoparametric lines in relation to the variable e^2_m. These lines can be seen in Figure 3, which illustrates isoparametric lines in relation to the variable e^2_m for balanced data, that is, $n_1 = n_2 = \frac{n}{2}$. Notice that the points that intercept the coordinated axes for balanced data are given by:

$$\begin{aligned} (0, e^2_{m_2}) &= (0, \frac{n_1+n_2}{n_2}\cdot e^2_m) = (0,\ 2\cdot e^2_m) \quad \text{(a)} \\ (e^2_{m_1}, 0) &= (\frac{n_1+n_2}{n_1}\cdot e^2_m, 0) = (2\cdot e^2_m,\ 0) \quad \text{(b)} \end{aligned} \tag{15}$$

It is interesting to observe that the point $(e^2_{m_1}, e^2_{m_2}) = (e^2_m, e^2_m)$ belongs to the equation of the line given by $e^2_{m_2} = \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_{m_1}$, independently of the chosen values for n_1 and n_2. The demonstration of that is simple enough, as it can be inferred from as follows:

$$\begin{aligned} e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_{m_1} \\ e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_m \\ e^2_{m_2} &= (\frac{n_1+n_2}{n_2} - \frac{n_1}{n_2})\cdot e^2_m \\ e^2_{m_2} &= (\frac{n_1+n_2-n_1}{n_2})\cdot e^2_m \\ e^2_{m_2} &= (\frac{n_2}{n_2})\cdot e^2_m \\ e^2_{m_2} &= e^2_m \end{aligned} \tag{16}$$

Hence, the point $(e^2_{m_1}, e^2_{m_2}) = (e^2_m, e^2_m)$ belongs to the line given by equation (10).

Figures 4 and 5 illustrate isoparametric lines, in relation to the variable e^2_m for non-balanced data, that is, $n_1 = \frac{n_2}{1000}$. Notice that the points that intercept the coordinate axis for balanced data are given by:

$$(0, e^2_{m_2}) = (0, \frac{n_1+n_2}{n_2} \cdot e^2_m) = (0, \frac{1001}{1000} \cdot e^2_m) \quad \text{(a)}$$
$$(e^2_{m_1}, 0) = (\frac{n_1+n_2}{n_1} \cdot e^2_m, 0) = (1001 \cdot e^2_m, 0) \quad \text{(b)} \qquad (17)$$

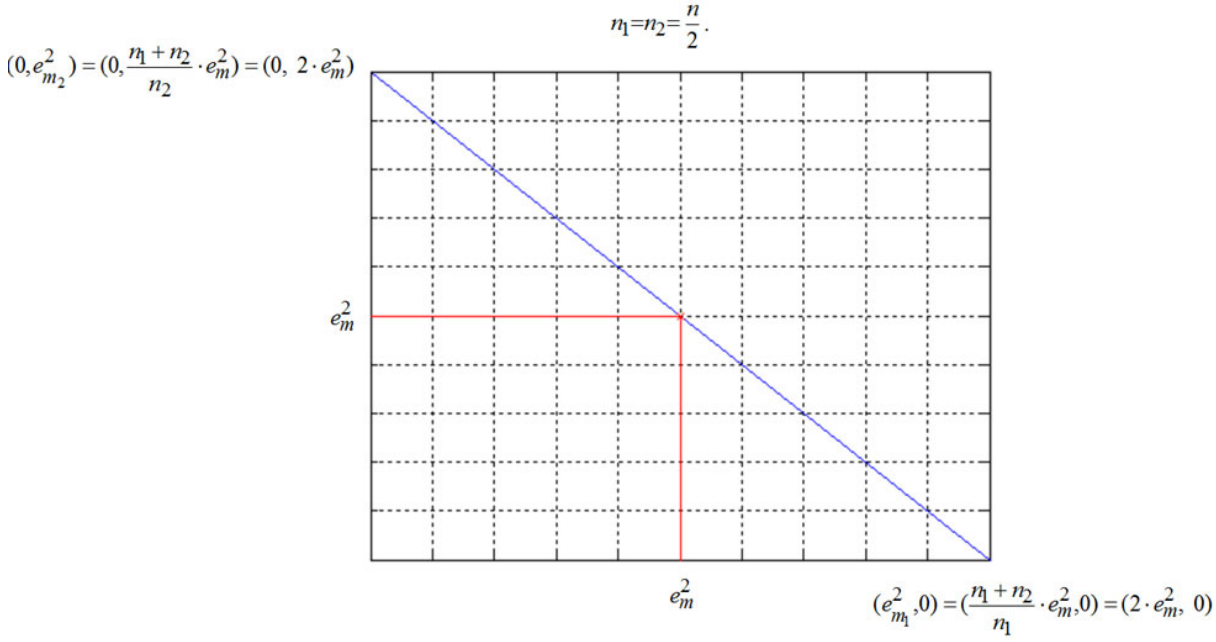

Figure 2. The graphical illustration of the equation of the line given by equation (10) for balanced data

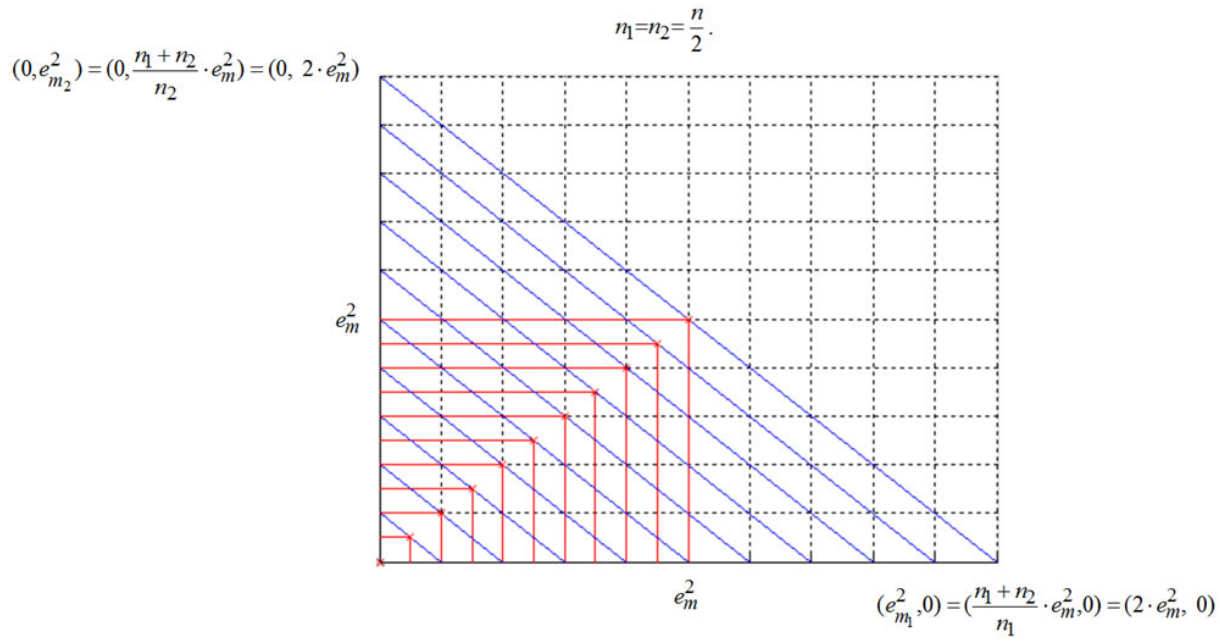

Figure 3. Isoparametric lines in relation to the variable e^2_m for balanced data

Demonstration: From the equation (14.a), the pair $(0, e^2_{m_2})$ is calculated as follows:

$$(0,e^2_{m_2}) = (0,\frac{n_1+n_2}{n_2}\cdot e^2_m) = (0,\frac{\frac{n_2}{1000}+n_2}{\frac{n_2}{1}}\cdot e^2_m)$$

$$= (0,\frac{\frac{n_2+1000\cdot n_2}{1000}}{\frac{n_2}{1}}\cdot e^2_m) = (0,\frac{n_2\cdot(1+1000)}{1000\cdot n_2}\cdot e^2_m) = (0,\frac{1001}{1000}\cdot e^2_m) \quad (18)$$

Equivalently, from the equation (14.a), the pair $(e^2_{m_1}, 0)$ is calculated as follows:

$$(e^2_{m_1},0) = (\frac{n_1+n_2}{n_1}\cdot e^2_m,0) = (\frac{n_1+1000n_1}{n_1}\cdot e^2_m,0) = (1001\cdot e^2_m,0) \quad (19)$$

Thus, Figures 4 and 5 present isoparametric lines for the non-balanced case. Notice that, for non-balanced data, the dispersion between $e^2_{m_1}$ and $e^2_{m_2}$ is much greater than for the balanced data case. In fact, for non-balanced data case this dispersion can reach the infinite.

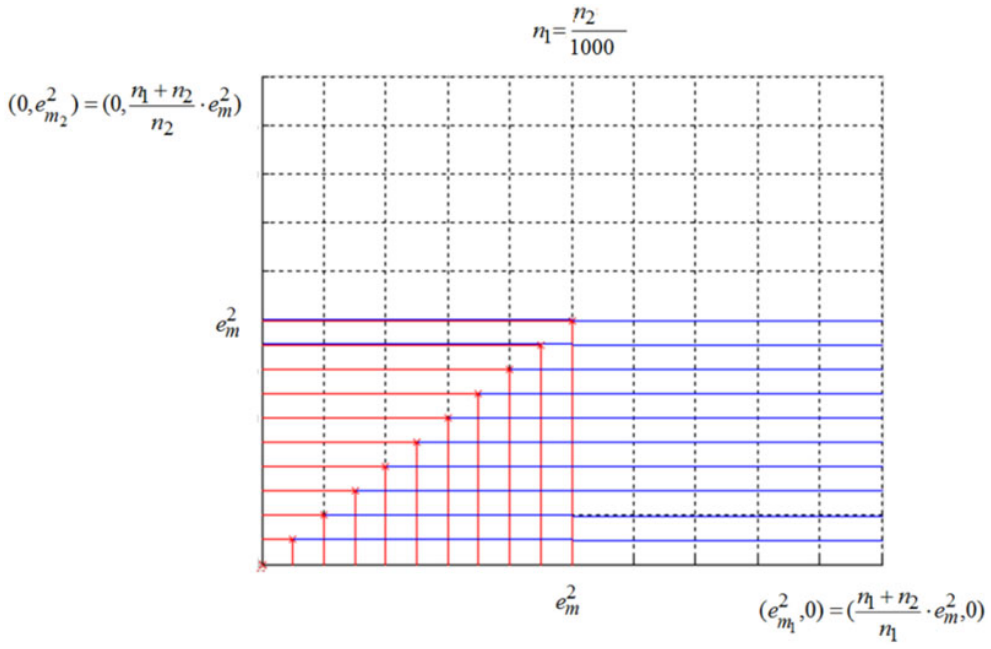

Figure 4. Isoparametric lines for non-balanced data, that is $n_1 = \frac{n_2}{1000}$

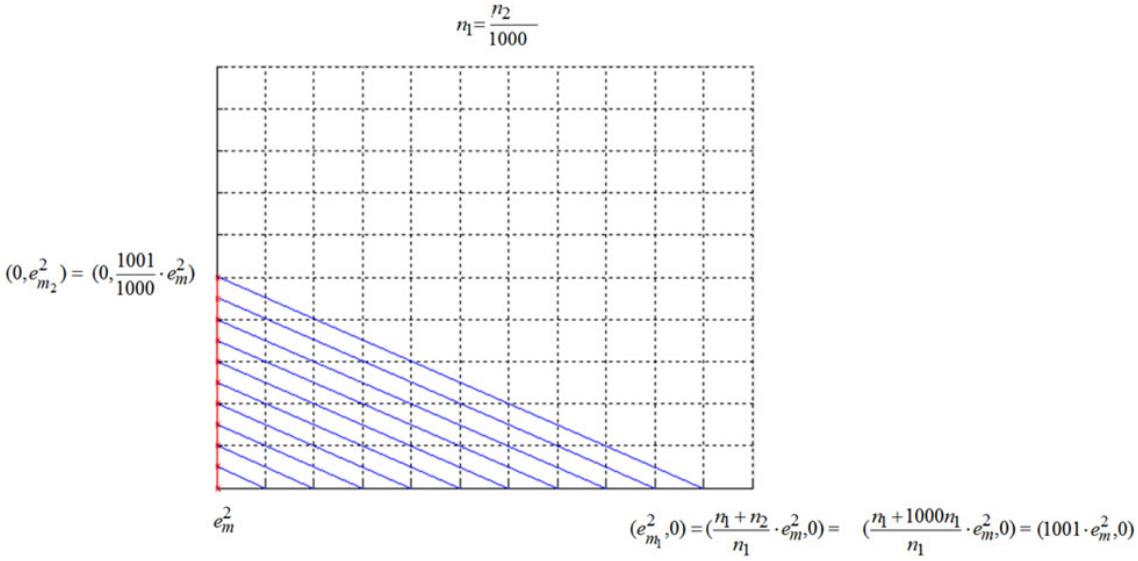

Figure 5. Isoparametric lines for non-balanced data, that is $n_1 = \frac{n_2}{1000}$

An important issue concerning the equation of the line given by equation (10) is the point where $e^2_{m_1} = 1$. For this point, in particular, one has:

$$\begin{aligned} e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot e^2_{m_1} \\ e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2}\cdot 1 \\ e^2_{m_2} &= \frac{n_1+n_2}{n_2}\cdot e^2_m - \frac{n_1}{n_2} \\ e^2_{m_2} &= \frac{n}{n_2}\cdot e^2_m - \frac{n_1}{n_2} \end{aligned} \tag{20}$$

Therefore, the point $(e^2_{m_1}, e^2_{m_2}) = (1, \frac{n}{n_2}\cdot e^2_m - \frac{n_1}{n_2})$ belongs to a line in Equation (10). Now, one has all needed information to elaborate a Monte Carlo algorithm to estimate the training behavior of a neural network in function of mean square error. For this, one needs to divide the simulation into three distinct parts, as follows:

(1) Analytical behavior of mean square error of perfectly balanced data, that is, $n_1 = n_2$;

(2) Analytical behavior of mean square error of extremely or completely non-balanced data $n_1 \ll n_2$;

(3) Analytical behavior of mean square error of intermediate data, which are *more or less balanced* or equivalently *more or less non-balanced*.

Balanced Data - based on empirical observations, one can state that for perfectly balanced data, the neural training converges to the point $(e^2_{m_1}, e^2_{m_2}) \cong (e^2_m, e^2_m)$ (see Figures 6, 26, and 27), that is, around the intersection of the line of the mean square errors and the bisector line of the coordinate axis. Thus, one can formulate the first simulation hypothesis.

H1: For perfectly balanced data, the neural training converges to $(e^2_{m_1}, e^2_{m_2}) \cong (e^2_m, e^2_m)$.

Justification for H1: One should notice that the ordered pair $(e^2_{m_1}, e^2_{m_2})$ must necessarily be contained in a point of the line present in Figure 6, and never out of it. As data are perfectly balanced, one expects that the chosen point for the neural training convergence might be around the point (e^2_m, e^2_m), that is, the mean point of the line of Figure 6.

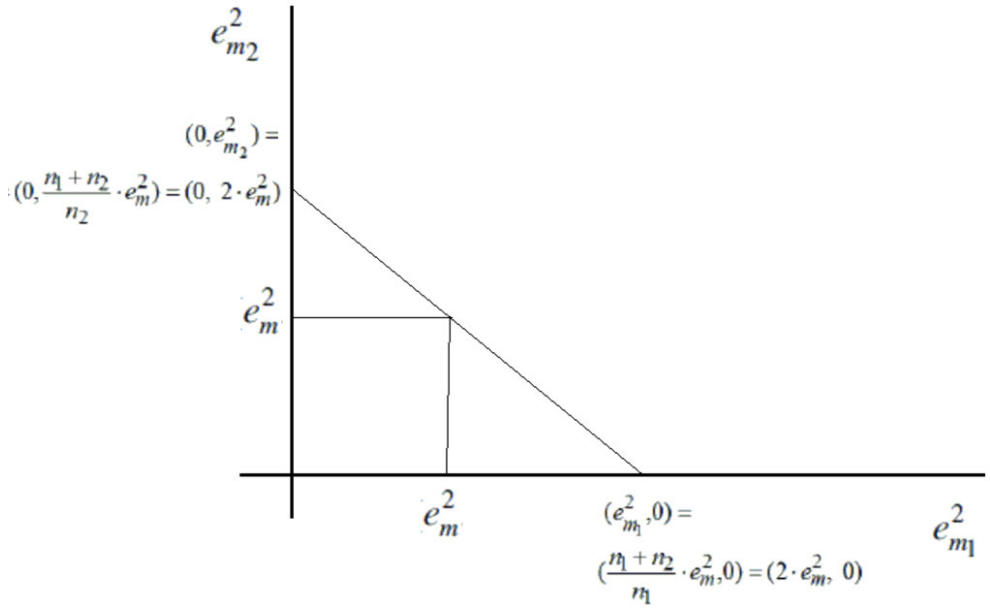

Figure 6. Analytical behavior of mean square errors among perfectly balanced data

Completely Non-Balanced Data - also based on empirical observations, one can state that for perfectly non-balanced data the neural training converges according to H2.

H2: For completely non-balanced data ($n_1 \lll n_2$), the neural training converges to H2 (see Figure 7).

If $\left(\frac{n_1+n_2}{n_1}\right)\cdot e^2_m \leq 1$, so, consider the convergence pair given by $(e^2_{m_1}, e^2_{m_2}) = \left(\left(\frac{n_1+n_2}{n_1}\right)\cdot e^2_m, 0\right)$ (in the worst case).

If $\left(\frac{n_1+n_2}{n_1}\right)\cdot e^2_m > 1$, consider the convergence pair given by $\left(1, \left(\frac{n_1+n_2}{n_2}\right)\cdot e^2_m - \left(\frac{n_1}{n_2}\right)\right)$ for $n_{12} = \frac{n_1}{n_2} < -\frac{e^2_m}{e^2_m - 1}$ and $e^2_m - 1 < 0$ (first existence condition) or $n_{12} = \frac{n_1}{n_2} = \frac{e^2_m}{e^2_m - 1}$ and $e^2_m - 1 \neq 0$ (second existence condition). These both existence conditions do not need to be filled. At the same time, only one of them should be satisfied in each simulation.

ISSN 1927-6974 E-ISSN 1927-6982

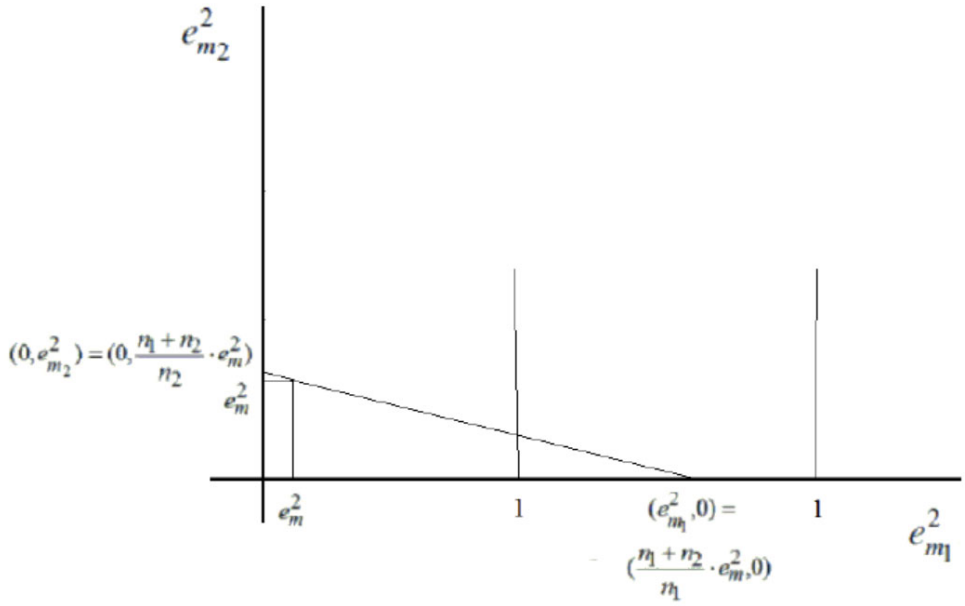

Figure 7. Analytical behavior of mean square errors among non-balanced data

Justification for H2: When any neural training is performed, after some training periods have elapsed, it is to be expected that mean square errors (e^2_m,$e^2_{m_1}$ and $e^2_{m_2}$) stay confined at any point between 0 and 1; In other words, it is warranted at least a minimum of learning. Thus, values greater than 1 and less than 0 for mean square errors are not allowed in Monte Carlo simulations, which are those one intends to perform in this article. Based on this, the first conditional for H2 states that if $e^2_{m_1} = \left(\frac{n_1+n_2}{n_1}\right) \cdot e^2_m$ is less than 1, the training algorithm is forced to converge to the convergence point given by $\left(\left(\frac{n_1+n_2}{n_1}\right) \cdot e^2_m, 0\right)$ (in the worst case). If $e^2_{m_1} = \left(\frac{n_1+n_2}{n_1}\right) \cdot e^2_m$ is greater than 1, the training algorithm will prefer the convergence point $\left(1, \left(\frac{n_1+n_2}{n_1}\right) \cdot e^2_m - \left(\frac{n_1}{n_2}\right)\right)$, because $e^2_{m_1}$ cannot be greater than 1, due to the minimum learning criteria (see Figure 7). In Figure 7, it is possible to notice that the point $\left(e^2_{m_1}, 0\right)$ can be located before the coordinated point (1,0) or after it. The previous argumentation took these two cases into account. Beyond this, notice that Figure 6 shows that for perfectly balanced data, the point (e^2_m, e^2_m) divides the mean square errors line into two equal parts. Nevertheless, for non-balanced data (see Figure 7), though the point (e^2_m, e^2_m) still belongs to the mean square errors line, this point always is in an asymmetrical position near the coordinated axes source.

The first and the second existence conditions are imposed to avoid negative values for $e^2_{m_2}$, because the means of mean square errors can never be negative. So, one should notice:

$$
\begin{aligned}
& e^2_{m_2} \geq 0 \\
& \left(\frac{n_1}{n_2}+1\right) \cdot e^2_m - \frac{n_1}{n_2} \cdot e^2_{m_1} \geq 0 \\
& \left(\frac{n_1}{n_2}+1\right) \cdot e^2_m - \frac{n_1}{n_2} \cdot 1 \geq 0 \\
& \left(n_{12}+1\right) \cdot e^2_m - n_{12} \cdot 1 \geq 0 \qquad \text{for} \qquad n_{12} = \frac{n_1}{n_2} \\
& \left(e^2_m - 1\right) \cdot n_{12} \geq -e^2_m
\end{aligned}
\tag{21}
$$

The last inequality equation of (21) is re-presented as follows:

$$
\left(e^2_m - 1\right) \cdot n_{12} \geq -e^2_m \tag{22}
$$

So, the inequality (22) can be unraveled in three cases:

$$
\begin{aligned}
& n_{12} > -\frac{e^2_m}{\left(e^2_m - 1\right)} \text{ if } \left(e^2_m - 1\right) > 0 \quad \text{(a)} \\
& n_{12} < -\frac{e^2_m}{\left(e^2_m - 1\right)} \text{ if } \left(e^2_m - 1\right) < 0 \quad \text{(b)} \\
& n_{12} = -\frac{e^2_m}{\left(e^2_m - 1\right)} \text{ if } \left(e^2_m - 1\right) \neq 0 \quad \text{(c)}
\end{aligned}
\tag{23}
$$

Values of $e^2_{m_2}$ that result in $(e^2_m - 1) = 0$ should be avoided, in order to avert problems of division by zero in equality (23.c). Values of $e^2_{m_2}$ that result in $(e^2_m - 1) > 0$ can be

disregard, because the most important simulations will be performed for values of e_m^2 in the interval $e_{m_2}^2$. This way, the most important inequality that interests is (23.b).

Partially Unbalanced Data - For intermediate data, one can choose to do a linear interpolation or an exponential interpolation on both extreme points indicated in H1 and H2. In this case, one should consider the following two situations:

If $\left(\frac{n_1+n_2}{n_1}\right) \cdot e_m^2 \leq 1$ so consider two interpolations, one interpolation on the points $\left(\left(\frac{n_1}{n_2}\right) = \lambda_\infty, e_{m_1}^2 = \left(1+\frac{n_2}{n_1}\right) \cdot e_m^2 = (1+\lambda_\infty) \cdot e_m^2\right)$ (maximum imbalance point) and $(\left(\frac{n_1}{n_2}\right) = 1, e_{m_1}^2 = e_m^2)$ (perfectly balancing point) to establish the value of $e_{m_1}^2$, and another linear interpolation on the line $e_{m_2}^2 = \frac{n_1+n_2}{n_2} \cdot e_m^2 - \frac{n_1}{n_2} \cdot e_{m_1}^2$ to determine the value of $e_{m_2}^2$. The first of these two interpolations for the determination of $e_{m_1}^2$ can be a linear, as well as an exponential one. The variable λ_∞ is an adjustment constant and a very great value where occurs a total imbalance.

If $\left(\frac{n_1+n_2}{n_1}\right) \cdot e_m^2 > 1$, so consider two interpolations, one interpolation on the points

$\left(\left(\frac{n_1}{n_2}\right) = \lambda_\infty, e_{m_1}^2 = 1\right)$ (maximum imbalance point) and $(\left(\frac{n_1}{n_2}\right) = 1, e_{m_1}^2 = e_m^2)$ (perfectly balancing point) to establish the value of $e_{m_1}^2$, and another linear interpolation on the line $e_{m_2}^2 = \frac{n_1+n_2}{n_2} \cdot e_m^2 - \frac{n_1}{n_2} \cdot e_{m_1}^2$ to determine the value of $e_{m_2}^2$. The first of these both interpolations for the determination of $e_{m_1}^2$ can be a linear, as well as an exponential one. In such a way, one can formulate H3 as follows:

H3: For partially non-balanced data, that is, $1 < \left(\frac{n_2}{n_1}\right) \equiv n_{21} < \lambda_\infty$, one has whenever:

(i) If $\left(\frac{n_1+n_2}{n_1}\right) \cdot e_m^2 \leq 1$, so consider the pair $\left(e_{m_1}^2, e_{m_2}^2\right)$, given by $e_{m_1}^2 = \left[\lambda_\infty \cdot \left(\frac{n_{21}-1}{\lambda_\infty - 1}\right)+1\right] e_m^2$ (linear) or $e_{m_1}^2 = e_m^2 \cdot n_{21}^{\left[\frac{\ln(1+\lambda_\infty)}{\ln \lambda_\infty}\right]}$ (exponential) and $e_{m_2}^2 = \frac{n_1+n_2}{n_2} \cdot e_m^2 - \frac{n_1}{n_2} \cdot e_{m_1}^2$ or $e_{m_2}^2 = \frac{1+n_{21}}{n_{21}} \cdot e_m^2 - \frac{1}{n_{21}} \cdot e_{m_1}^2$.

(ii) If $\left(\frac{n_1+n_2}{n_1}\right) \cdot e_m^2 > 1$, so consider the pair $(e_{m_1}^2, e_{m_2}^2)$, given by $e_{m_1}^2 = \left(\frac{n_{21}-1}{\lambda_\infty - 1}\right) \cdot \left(1-e_m^2\right) + e_m^2$ (linear) and $e_{m_2}^2 = \frac{n_1+n_2}{n_2} \cdot e_m^2 - \frac{n_1}{n_2} \cdot e_{m_1}^2$ or $e_{m_2}^2 = \frac{1+n_{21}}{n_{21}} \cdot e_m^2 - \frac{1}{n_{21}} \cdot e_{m_1}^2$.

For

$$n_{12} = \frac{n_1}{n_2} > -\frac{e_m^2}{e_m^2 - e_{m_1}^2} \text{ and } e_m^2 - e_{m_1}^2 > 0 \text{ (first existence condition)} \quad \text{(a)}$$

$$n_{12} = \frac{n_1}{n_2} < -\frac{e_m^2}{e_m^2 - e_{m_1}^2} \text{ and } e_m^2 - e_{m_1}^2 < 0 \text{ (second existence condition)} \quad \text{(b)}$$

$$n_{12} = \frac{n_1}{n_2} = -\frac{e_m^2}{e_m^2 - e_{m_1}^2} \text{ and } e_m^2 - e_{m_1}^2 \neq 0 \text{ (third existence condition)} \quad \text{(c)}$$

(24)

where λ_∞ is a great value where a total imbalance of data occurs (it is an adjustment constant) and $1 < n_{21} = \left(\frac{n_2}{n_1}\right) < \lambda_\infty$. As it can be noticed, there are two allowed interpolations to determine $e_{m_1}^2$: one is linear; and the other is exponential. They practically produce the same outcome, and therefore, it does not matter which of them ones uses in practice. Again, there are three existence conditions for the mean square error $e_{m_2}^2$. Nevertheless, only one of them needs to be satisfied at the same time for each Monte Carlo simulation.

A question remains: How can one find the expressions for the linear or exponential interpolation? In order to answer this question, see Figure 8, which considers the specific case of condition (i) in H3.

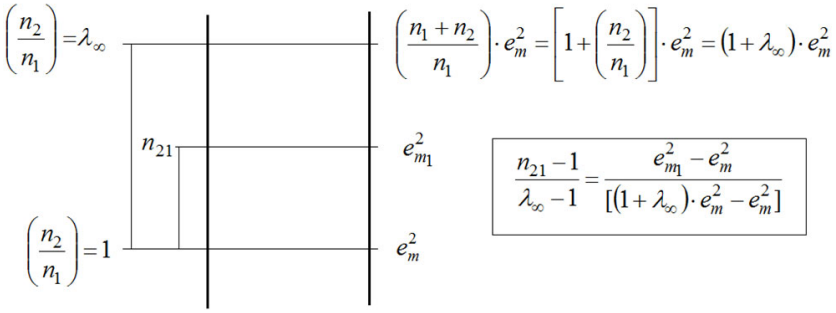

Figure 8. The linear interpolation for the determination of $e_{m_1}^2$, for partially non-balanced data

As it can be noticed in Figure 8, a linear interpolation between the points $\left(\left(\frac{n_1}{n_2}\right) = \lambda_\infty, e^2_{m_1} = \left(1 + \frac{n_2}{n_1}\right) \cdot e^2_m = (1 + \lambda_\infty) \cdot e^2_m\right)$ and $(\left(\frac{n_1}{n_2}\right) = 1, e^2_{m_1} = e^2_m)$ produces the following line equation:

$$\frac{n_{21} - 1}{\lambda_\infty - 1} = \frac{e^2_{m_1} - e^2_m}{[(1 + \lambda_\infty) \cdot e^2_m - e^2_m]} \tag{25}$$

By isolating the variable $e^2_{m_1}$ of equation (25), it results in:

$$e^2_{m_1} = \left[\lambda_\infty \cdot \left(\frac{n_{21} - 1}{\lambda_\infty - 1}\right) + 1\right] e^2_m \tag{26}$$

Equation (26) is the same as the one, which appears in condition (i) of H3. Similarly, the exponential interpolation uses the same extreme points that were used in linear interpolation. However, instead of using an equation of a line, one uses the equation of an exponential. So, a possible exponential interpolation would be:

$$e^2_{m_1} = p \cdot n^q_{21} \tag{27}$$

By subtracting the Napierian logarithm in both sides of equation (27), one has:

$$e^2_{m_1} = \ln p + q \cdot \ln n_{21} \tag{28}$$

where p and q are adjustment parameters. By substituting both extreme points $(\lambda_\infty, (1 + \lambda_\infty) \cdot e^2_m)$ and $(1, e^2_m)$ in equation (28), it results in the following linear system:

$$\begin{Bmatrix} \ln e^2_m \\ \ln(1 + \lambda_\infty) e^2_m \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & \ln \lambda_\infty \end{bmatrix} \cdot \begin{Bmatrix} \ln p \\ q \end{Bmatrix} \tag{29}$$

By solving (29) linear system, one has:

$$\begin{aligned} p &= e^2_m \quad \text{(a)} \\ q &= \frac{\ln(1 + \lambda_\infty)}{\ln \lambda_\infty} \quad \text{(b)} \end{aligned} \tag{30}$$

This way, the exponential interpolation equation results in:

$$e^2_{m_1} = e^2_m \cdot n_{21}^{\left(\frac{\ln(1+\lambda_\infty)}{\ln \lambda_\infty}\right)} \tag{31}$$

Analogously to the interpolation that appears in condition (ii) of H3, the following linear interpolation can be obtained:

$$\frac{n_{21} - 1}{\lambda_\infty - 1} = \frac{e^2_{m_1} - e^2_m}{1 - e^2_m} \tag{32}$$

By isolating the variable $e^2_{m_1}$ of equation (32), it results in:

$$e^2_{m_1} = \left(\frac{n_{21} - 1}{\lambda_\infty - 1}\right) \cdot \left(1 - e^2_m\right) + e^2_m \tag{33}$$

Equation (33) is the same as the one that appears in condition (ii) of H3.

Remark - For partially non-balanced points, it is not necessary to consider the case $e^2_{m_1} < e^2_m$, because this will never happen, if one takes care to consider only values less than 1 for e^2_m, that is always $e^2_m \leq 1$. Thus, the first existence condition for partially non-balanced data is not needed.

To finalize this section, an important result can be obtained by calculating the limit for $n_1 = 1$ and $n_2 \to \infty$ in equation (9.a). Using L'Hôpital's well-known rule to avoid indetermination of the type ∞/∞, we have:

$$\begin{aligned} &\lim_{n_2 \to \infty} e^2_m = \lim_{n_2 \to \infty} \frac{e^2_{m_1}}{1 + n_2} + \lim_{n_2 \to \infty} \frac{n_2}{1 + n_2} \cdot e^2_{m_2} \quad \text{(a)} \\ &\lim_{n_2 \to \infty} e^2_m = 0 + \lim_{n_2 \to \infty} e^2_{m_2} = e^2_{m_2} \quad \text{(b)} \end{aligned} \tag{34}$$

Equation (34.b) simply states that when there is only one fraud, mixed with an infinity of non-frauds, the value of e^2_m will only depend on $e^2_{m_2}$, independently of any value, which is for $e^2_{m_1}$, zero or infinite. Put another way, for this extreme case of imbalance, the neural network completely loses its ability to learn the fraud.

4. THE MONTE CARLO PROPOSED ALGORITHM

Based upon the previous sections and also on the mathematic tooling presented up to here, it is possible to propose a Monte Carlo algorithm to preview the theoretical behavior of neural training errors between balanced and non-balanced data for a typical non-linear classification problem of training patterns. It is assumed the particular case of having only two classes: an output equal to zero or an output equal to one. The proposed algorithm can be divided into three parts:

- The Monte Carlo algorithm for perfectly balanced data (see Figure 9);
- The Monte Carlo algorithm for partially non-balanced data (see Figures 10 and 11); and
- The Monte Carlo algorithm for completely non-balanced data (see Figures 12 and 13).

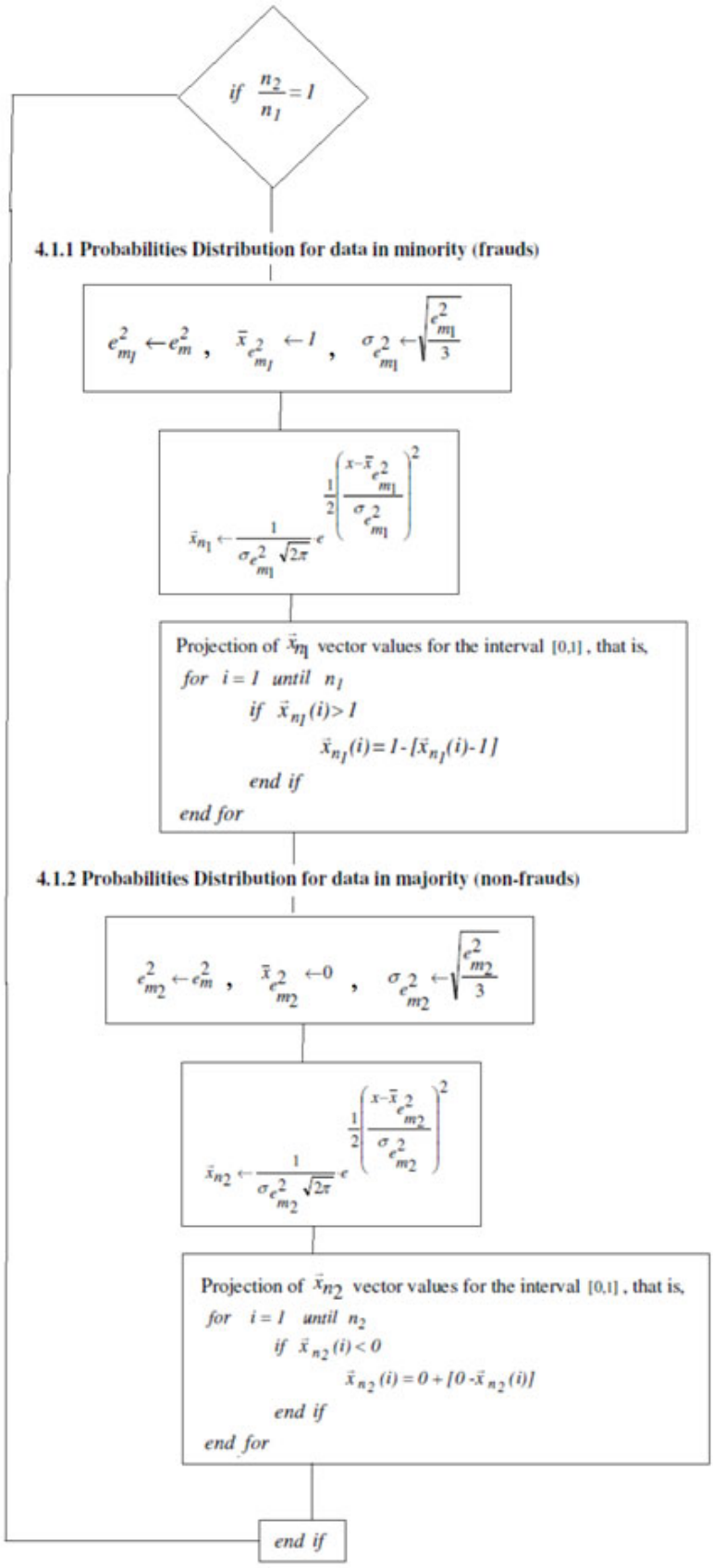

Figure 9. The Monte Carlo algorithm for perfect balanced data

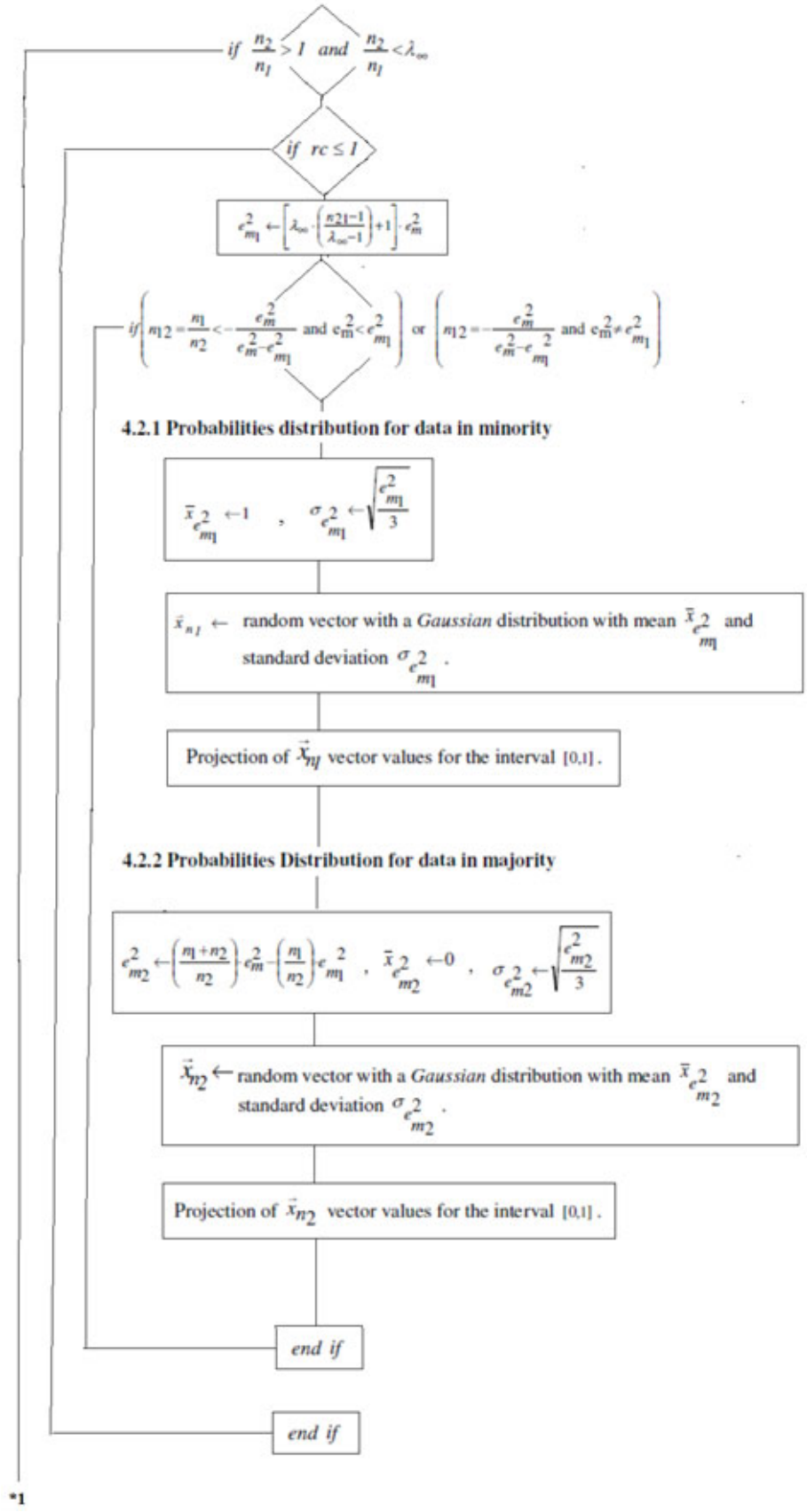

Figure 10. The Monte Carlo algorithm 4.2 for partially non-balanced data (Part I)

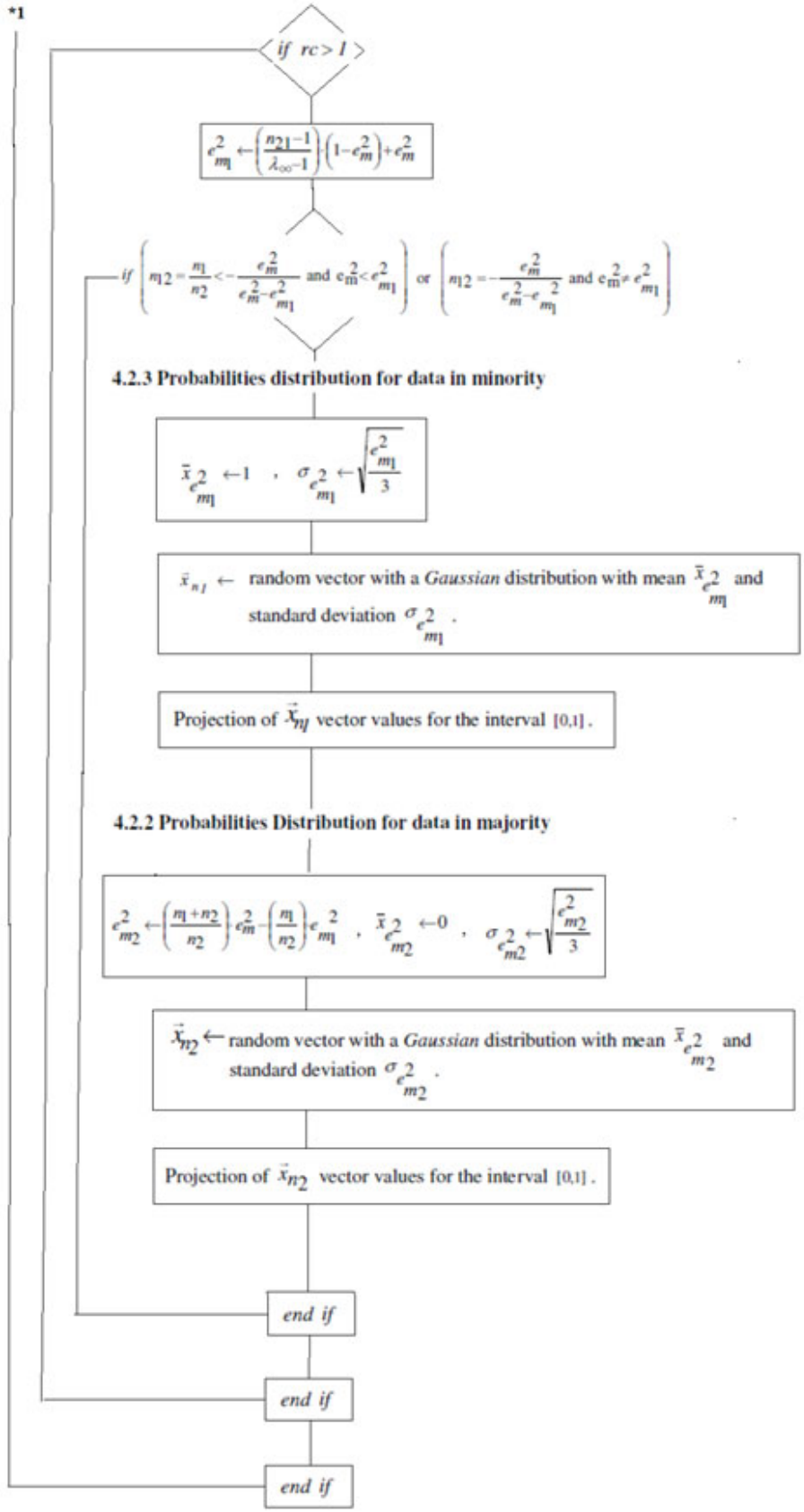

Figure 11. The Monte Carlo algorithm 4.2 for partially non-balanced data (Part II)

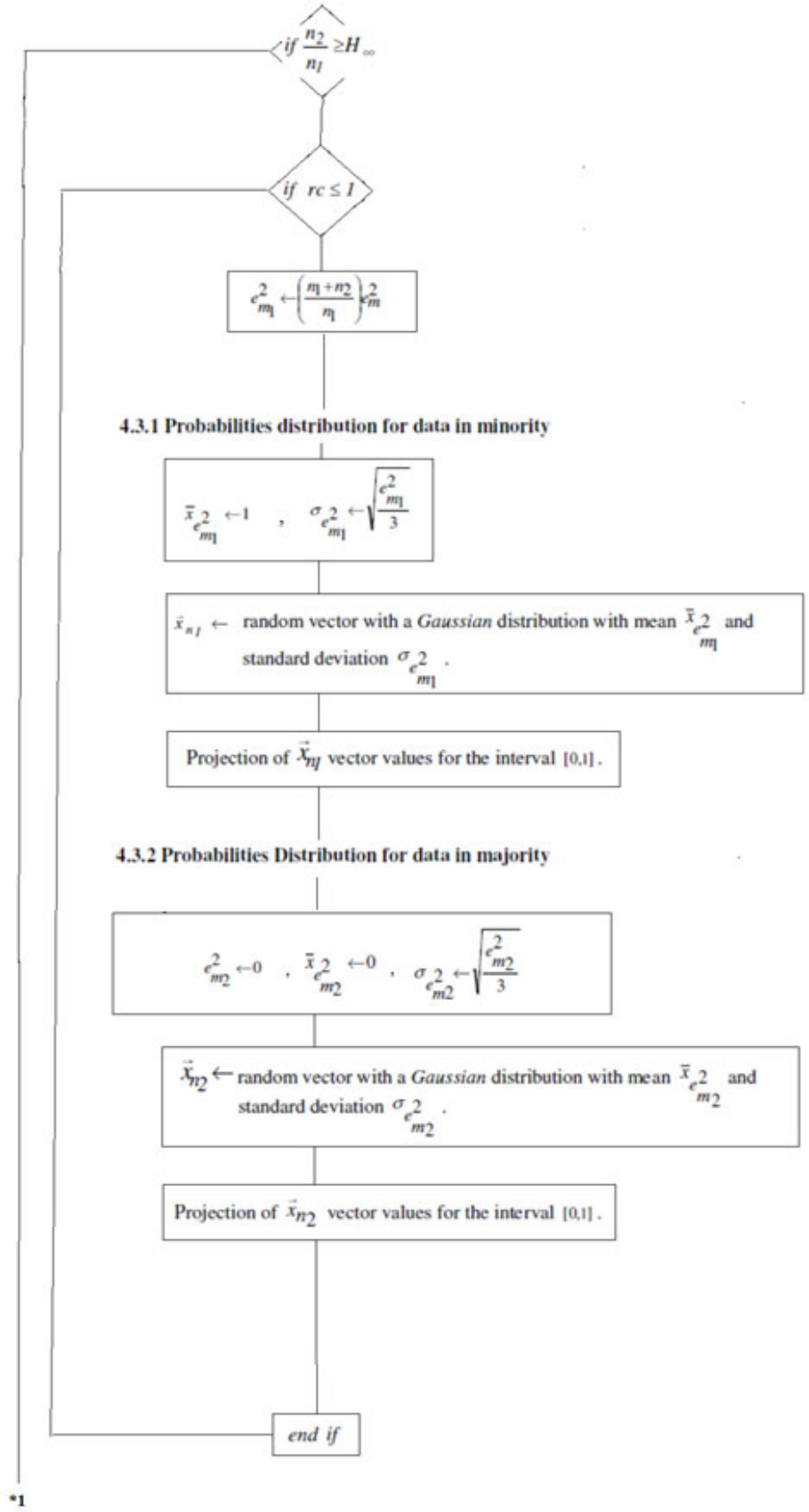

Figure 12. The Monte Carlo algorithm 4.3 for completely non-balanced data (Part I)

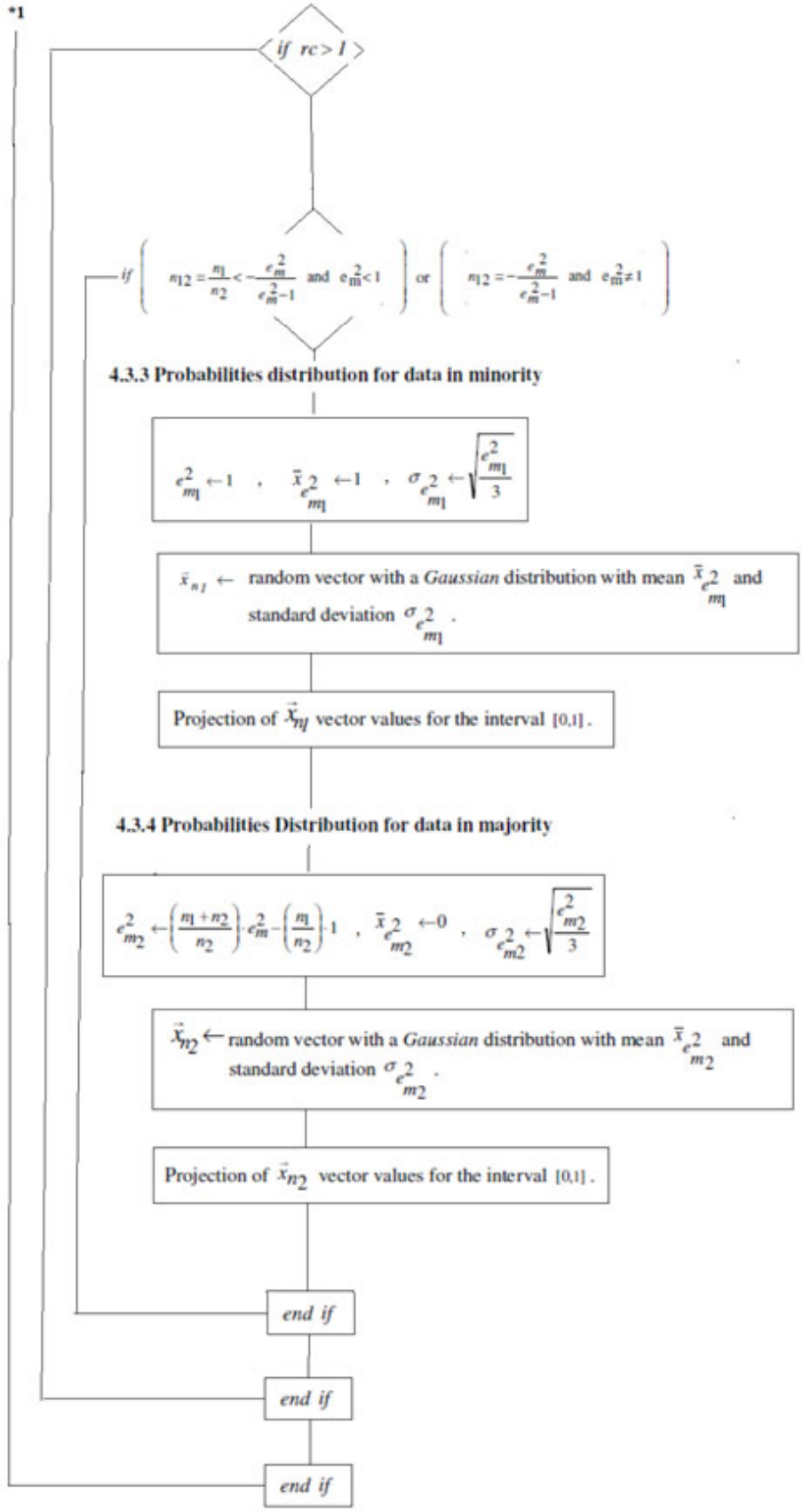

Figure 13. The Monte Carlo algorithm 4.3 for completely non-balanced data (Part II)

Nevertheless, before presenting these three partial algorithms, the input and output variables of the proposed algorithms are established as follows.

1. Input Variables

$e_m^2 \leftarrow [0,1]$ is the global mean square

$n_1 \leftarrow$ is the total number of data in the minority

$n_2 \leftarrow$ is the total number of data in the majority

$n \leftarrow n_1 + n_2$ is the total number of training patterns

$\lambda_\infty \leftarrow \left(\frac{n_2}{n_1}\right)_{terco} > 1$ is the theoretical relation from where a total imbalance occurs (high value adjustment constant)

$n_{21} \leftarrow \frac{n_2}{n_1}$ should be a value in the interval $[1, \lambda_\infty]$

$n_{12} \leftarrow \frac{n_1}{n_2}$

$rc \leftarrow \left(\frac{n_1 + n_2}{n_1}\right) \cdot e_m^2$

2. Output Variables

$e_{m_1}^2 \leftarrow$ is the mean square error of data in minority (frauds)

$e_{m_2}^2 \leftarrow$ is the mean square error of data in majority (non-frauds)

In fact, outputs are *Gaussian* curves with well-established means and standard deviation, that is, $\overline{x}_{e_{m_1}^2}$, $\sigma_{e_{m_1}^2}$, $\overline{x}_{e_{m_2}^2}$ and $\sigma_{e_{m_2}^2}$. The random vectors that obey these *Gaussian* distributions are respectively $\vec{x}_{n_1}$ and $\vec{x}_{n_2}$. The dimension of $\vec{x}_{n_1}$ is n_1 and the dimension of $\vec{x}_{n_2}$ is n_2. The random vectors $\vec{x}_{n_1}$ and $\vec{x}_{n2}$ are effectively the algorithm output.

The same rule is always valid for the three proposed algorithms, that is, one should establish: the mean square error $e_{m_1}^2$; the mean value of the dispersion $\overline{x}_{e_{m_1}^2}$; the standard deviation of $\sigma_{e_{m_1}^2}$ dispersion; the normal random vector $\vec{x}_{n_1}$; and the projection of $\vec{x}_{n_1}$ vector values inside the interval $[0,1]$ for data in minority. The value of $\sigma_{e_{m_1}^2}$ standard deviation is established by the expression $\sqrt{e_{m_1}^2/3}$, because this warrants the *Gaussian* random vector $\vec{x}_{n_1}$ has all its values limited in the interval $[1-e_{m_1},1]$ with 99.73% of sureness. This is repeated for data in majority. In order to follow all the progress of this algorithm, it is necessary an attentive reading of section 3 of this paper.

5. The numerical implementation of Monte Carlo algorithm

The numerical outcomes presented in this section were numerically obtained from the MATLAB application. The developed algorithm was exactly the one described in detail in section 4 of this paper. A total of 11 numerical simulations were performed (see Figures from 14 to 24). All of these simulations were performed with $n_2 = 650,000$ and $\lambda_\infty = 26$. If one allows n_1, varying from 650,000 to 25,000, it was possible to vary the value of n_{21}, as presented in Tables 1 and 2. In most of simulations, the values of $\lambda_\infty = 26$ and $e_m = \sqrt{e_m^2}$ were respectively established in 0.1000 and 0.3162. Nevertheless, these values were changed in Figures from 22 to 24.

As all n_1 values were altered, so the rc values also suffered meaningful variations as presented in Tables 1 and 2. The value of e_m^2 was strategically chosen equal to 0.1000, because by having these values in the case of balanced data, the training classification of patterns with zero or one output is satisfactorily obtained as presented in Figure 14. This is a frontier value, because values a little greater than that improve the classification of patterns just a little bit, and values lesser than that meaningfully injure the classification of patterns. What one intends here is to vary the value of n_{21}, more specifically, to increase it from 1 to 26 and check if the classification of training patterns improves or worsens with the imbalance of n_{21} for this specific frontier value adopted for e_m^2.

Table 1. Simulation of Numerical Data

	Figure 14	Figure 15	Figure 16	Figure 17	Figure 18	Figure 19
n_1	650,000	550,000	450,000	350,000	250,000	100,000
n_2	650,000	650,000	650,000	650,000	650,000	650,000
n_{21}	1.0000	1.1818	1.4444	1.8571	2.6000	6.5000
e_m^2	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
$e_m = \sqrt{e_m^2}$	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162
rc	0.2000	0.2182	0.24444	0.2857	0.3600	0.7500

By analyzing Figures from 14 to 18, it is verified that for an unbalance up to $n_{21} = 2.60000$, the classification of patterns does not worsen or improve. However, though a simple visual inspection from Figures 14 to 18, it is verified that a meaningful imbalance occurs in the classification of patterns. But, by observing Figures from 19 to 21, it is noticed that with the imbalance for n_{21}, varying between 6.5 and 26.000, the classification of patterns meaningfully worsens.

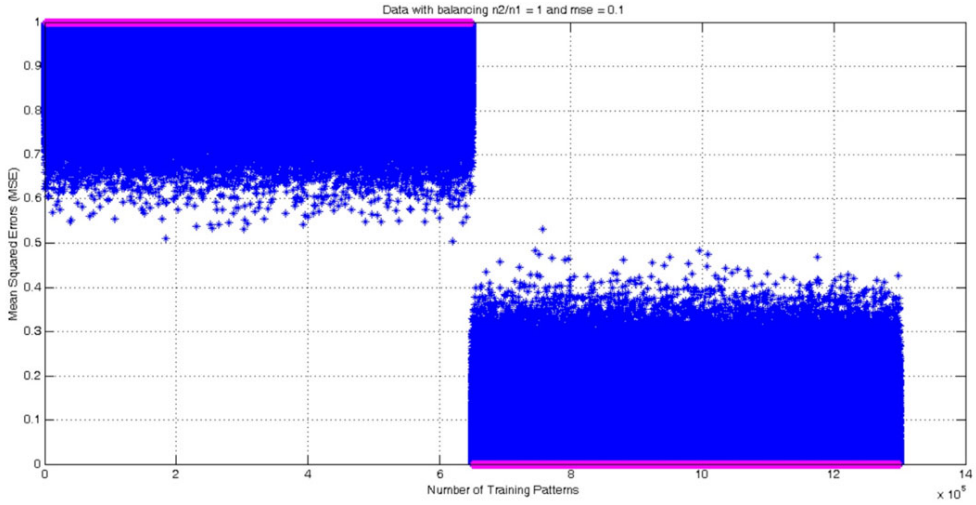

Figure 14. The Monte Carlo simulation for perfectly balanced data, that is $n_{21} = 1.0000$ and $rc = 0.2000$

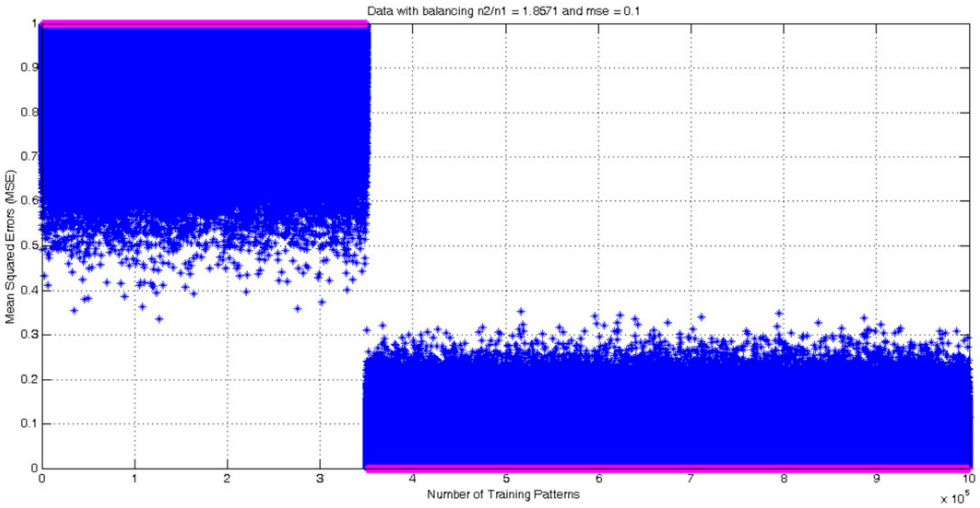

Figure 15. The Monte Carlo simulation for partially balanced data, with $n_{21} = 1.1818$ and $rc = 0.2182$

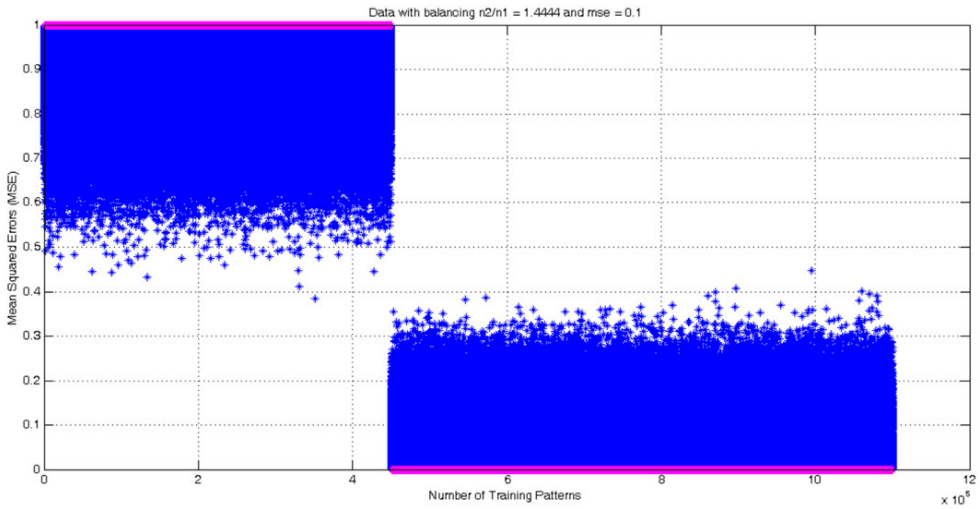

Figure 16. The Monte Carlo simulation for partially balanced data, with $n_{21} = 1.4444$ and $rc = 0.2444$

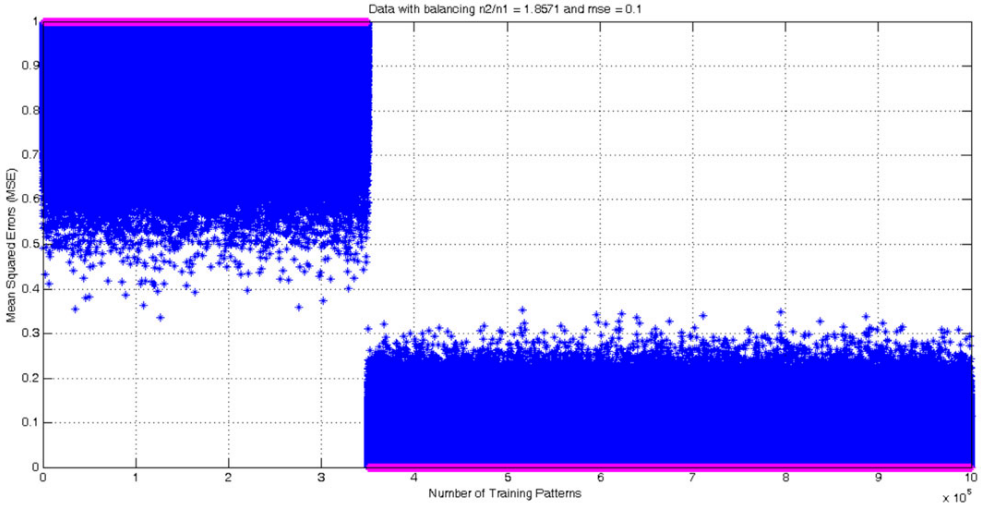

Figure 17. The Monte Carlo simulation for partially balanced data, with $n_{21} = 1.8571$ and $rc = 0.2857$

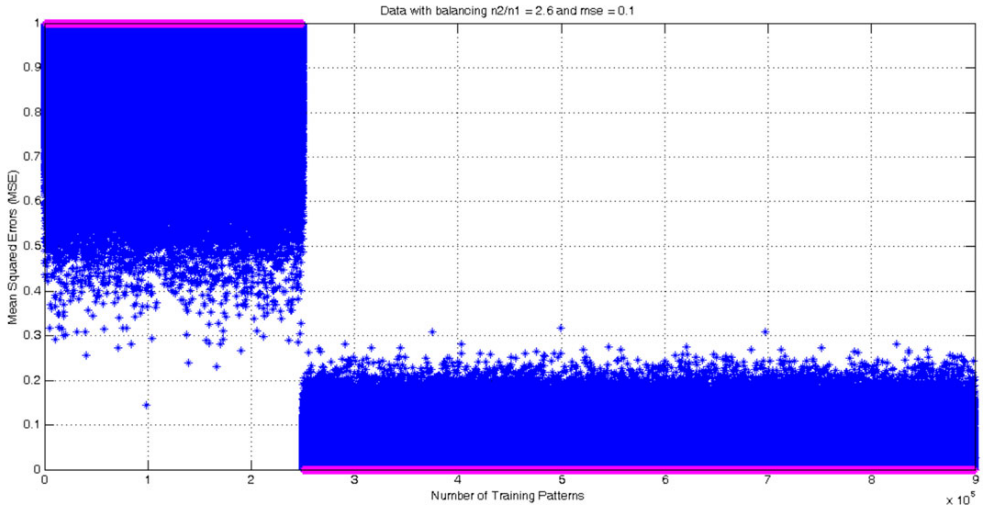

Figure 18. The Monte Carlo simulation for partially balanced data, with $n_{21} = 2.6000$ and $rc = 0.3600$

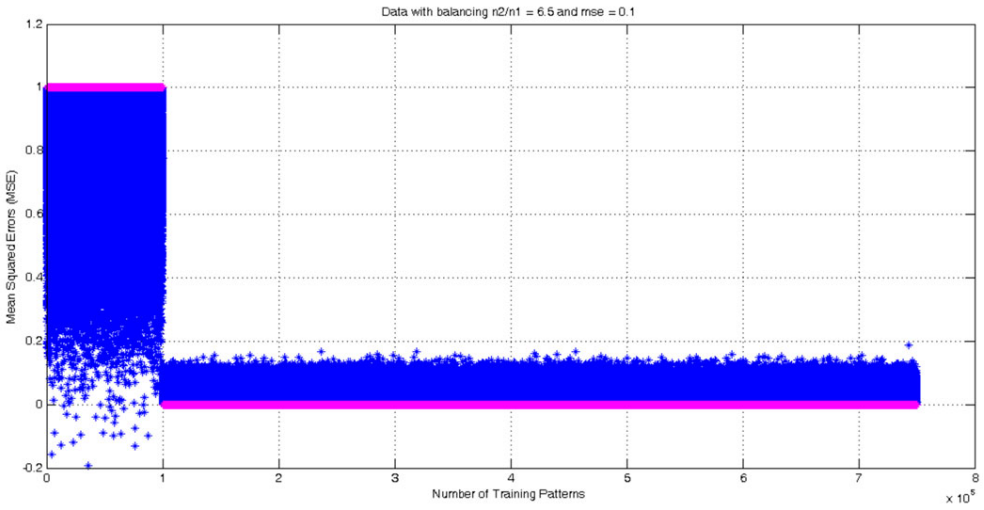

Figure 19. The Monte Carlo simulation for partially balanced data, with $n_{21} = 6.5000$ and $rc = 0.7500$

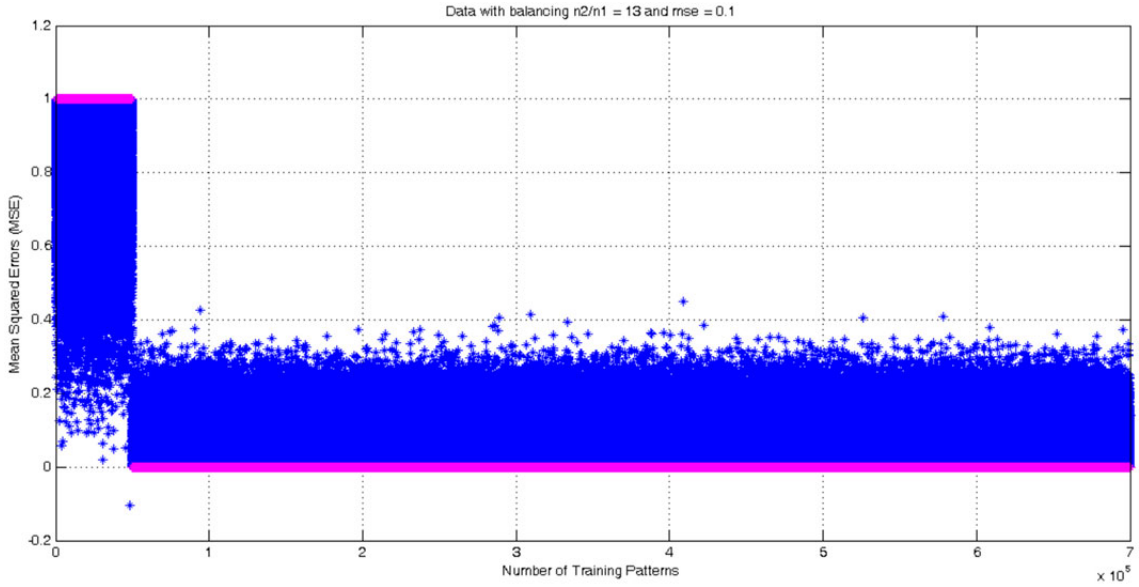

Figure 20. The Monte Carlo simulation for partially balanced data, with $n_{21} = 13.0000$ and $rc = 1.4000$

In the first analysis, it is verified that unbalanced data are more difficult of being learned than balanced data, because the former require a mean square error of learning that is much lesser than the values of the mean square errors of the balanced data to reach the same precision.

From Figures 21 and 22, it is noticed a little abrupt variation in the presented outcomes of one Figure in relation to another. This is due to the fact that the rc values of Figure 21 are greater than one and those of Figure 22 are lesser than 1 (see Table 2).

Table 2. Simulation of Numerical Data

	Figure 20	**Figure 21**	**Figure 22**	**Figure 23**	**Figure 24**
n_1	50,000	25,000	25,000	25,000	650,000
n_2	650,000	650,000	650,000	650,000	650,000
n_{21}	13.000	26.0000	26.0000	26.0000	1.0000
e_m^2	0.1000	0.1000	0.0200	0.0050	0.0050
$e_m = \sqrt{e_m^2}$	0.3162	0.3162	0.1414	0.0707	0.0707
rc	1.4000	2.7000	0.5400	0.1350	0.0100

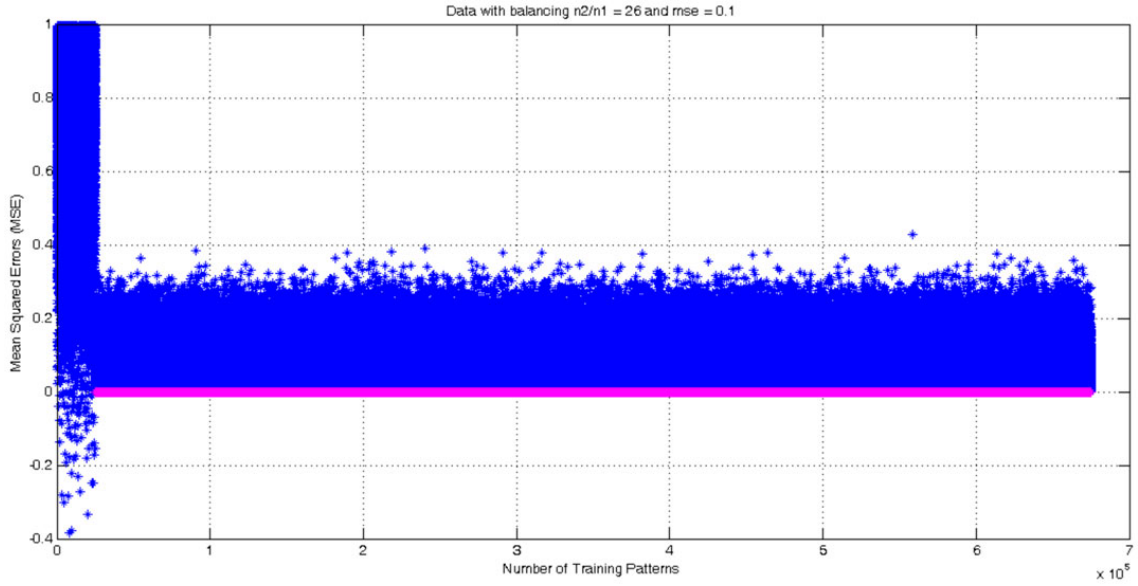

Figure 21. The Monte Carlo simulation for partially balanced data, with $n_{21} = 26.0000$ and $rc = 2.7000$

As it can be seen in algorithms 4.2 and 4.3, this causes a little variation in the calculation of mean square errors, because the variable rc is responsible for altering the mathematical formula that estimates the mean square errors of balanced and non-balanced data. However, as the presented model is an abstraction of reality, this does not meaningfully damage the numerical outcomes presented by Figures 21 and 22.

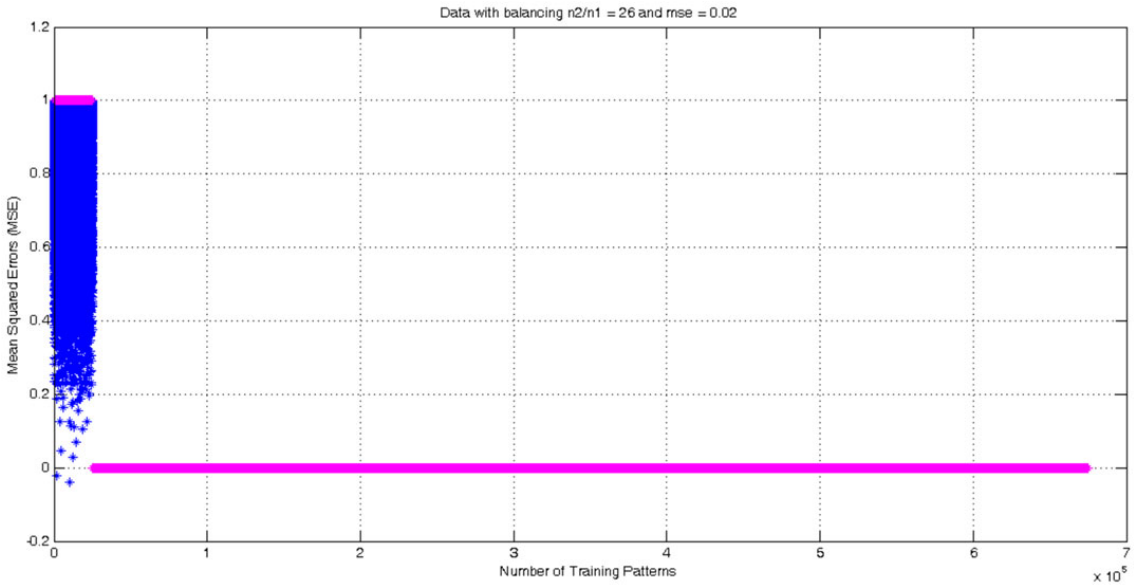

Figure 22. The Monte Carlo simulation for partially balanced data, with $n_{21} = 26.0000$ and $rc = 0.5400$

Figures 22 and 23 present the numerical outcomes for completely non-balanced data, as suggested by algorithm 4.3. This algorithm foresees a mean square error for abundant data that is equal to zero with a variance that is also equal to zero. Nevertheless, it is verified that in real world problem there will always be a little non-zero noise. However, the presence of this noise would just worsen the classification, which is already not good.

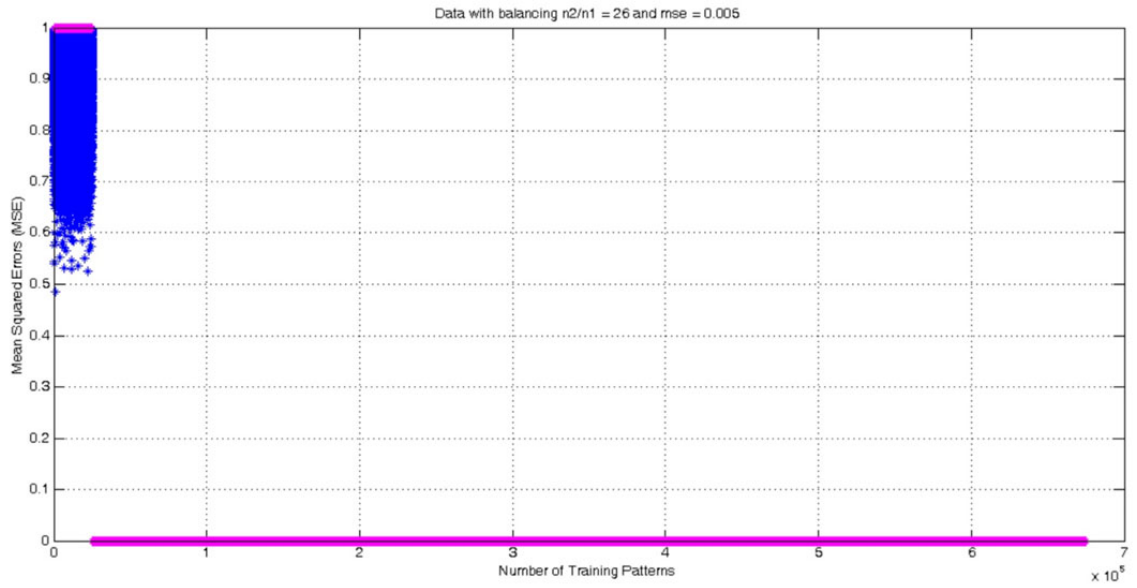

Figure 23. The Monte Carlo simulation for partially balanced data, with $n_{21} = 26.0000$ and $rc = 0.1350$

Figure 24 presents the numerical outcomes of perfectly balanced data in comparison to completely non-balanced data from Figure 23. The same value for e_m^2 was used in both of these simulations, 0.0050. Notice that the numerical behavior in both presented cases are quite different from each other. Based upon all the numerical simulations presented in this section, one can verify that, in average, balanced data are easier of being learned than non-balanced data. The balancing of data suggested here is the same as the one mentioned in section 5.1 of this paper, that is, to multiply all data in minority by a factor greater than one, in such a way that the amount of data in minority is very close to the amount of data in the majority. Evidently, this artifice causes several repetitions of the same data.

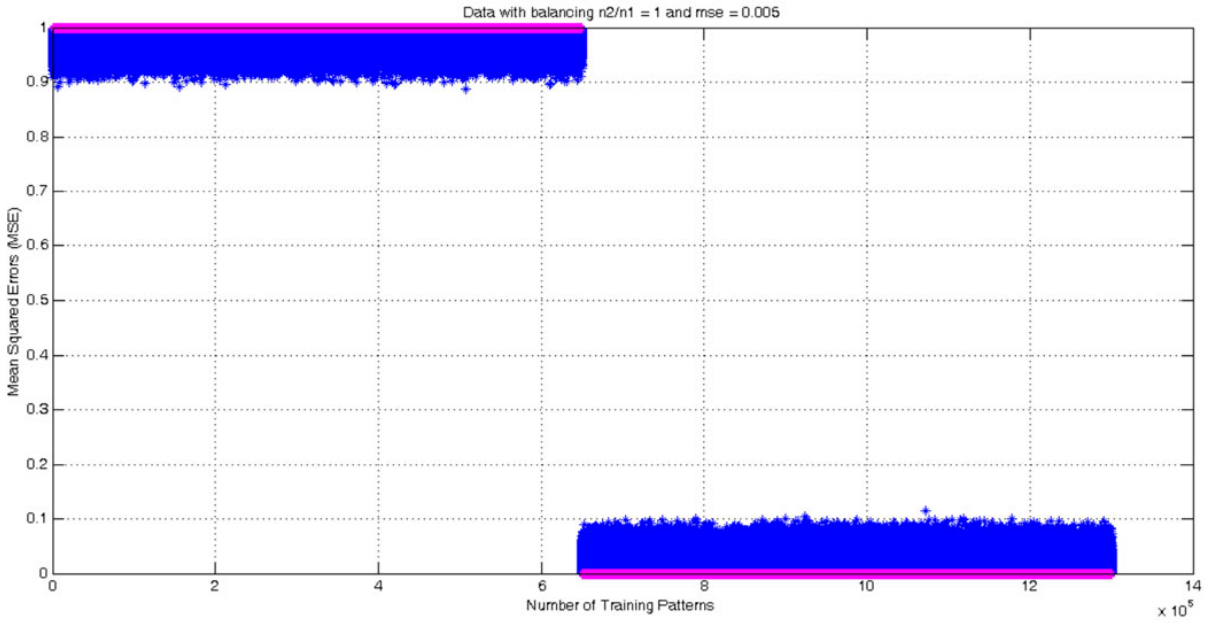

Figure 24. The Monte Carlo simulation for partially balanced data, with $n_{21} = 1.0000$ and $rc = 0.0100$

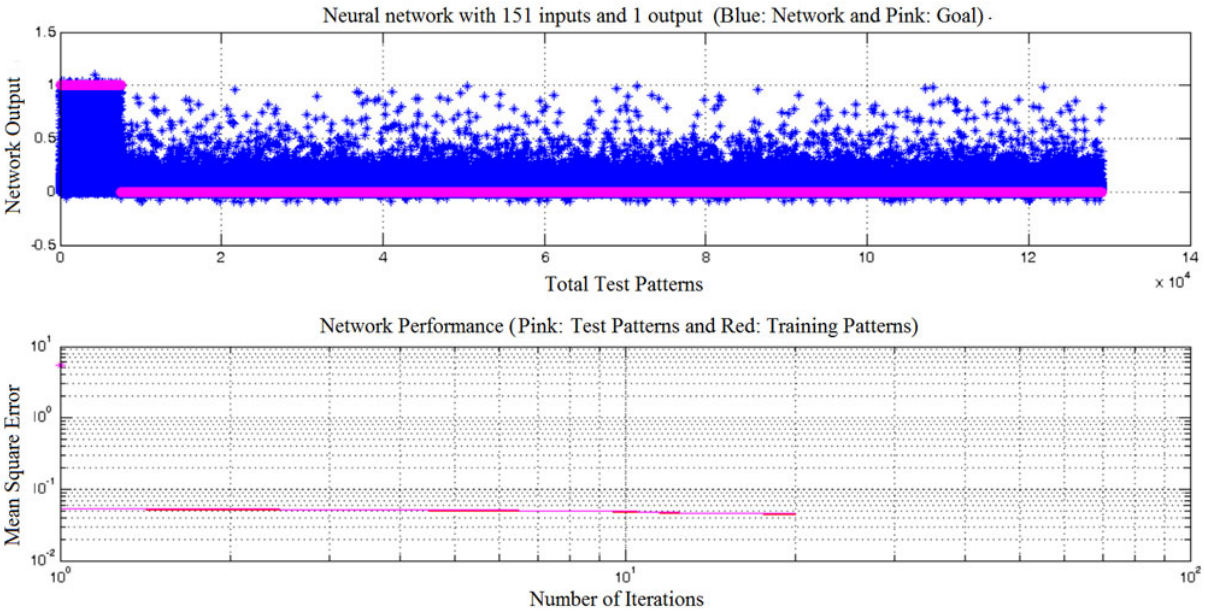

Figure 25. The Neural Training with non-balanced data

Experimental results obtained by real neural training

Figures 25 and 26 show a typical training where an MLP network with about 650 thousand non-balanced training patterns is used. This MLP application has used in this training 613 thousand outputs equal to zero and only 37 thousand outputs equal to one. In this case, eighty per cent of patterns were set apart for training and twenty percent for testing. Figures 25, 26, and 27 present the testing patterns only. Figures 25 and 26 represent real world data with credit card transactions performed in Brazil, where frauds (the minority of data) have outputs equal to one and non-frauds (the majority of data) have outputs equal to zero. Figure 27 shows the training with balanced data. From Figures 25, 26, and 27, one can realize that for the balanced data (see Figure 27) frauds and non-frauds tend to separate themselves in a uniform way. However, for non-balanced data (see Figures 25 and 26), the minority data tend to be classified as majority data, that is, in an unsuitable way. This paper showed that due to this phenomenon, non-balanced data are more difficult of being trained than balanced data.

It should be noticed that classifications obtained from MLP networks in Figures 25, 26, and 27 are not very good. Nevertheless, discovering a neural training method to improve these classifications is not the aim of this paper. However, it is important to say that the authors of this paper are currently trying to improve these classifications by using Neural Networks with *Deep Learning architecture on Tensorflow* framework. But this is an issue for another paper. What was intended here was to develop a model based in Monte Carlo method to theoretically preview and estimate how the decay of mean squared error of neural networks occurs when there is balancing and non-balancing data.

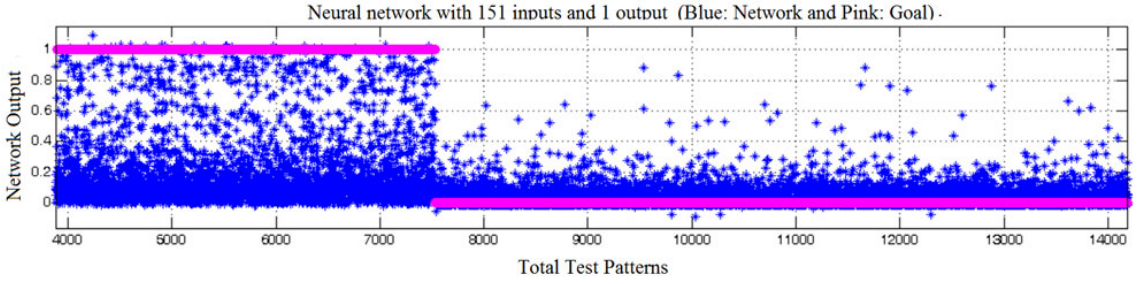

Figure 26. The same graphic of Figure 25, but amplified

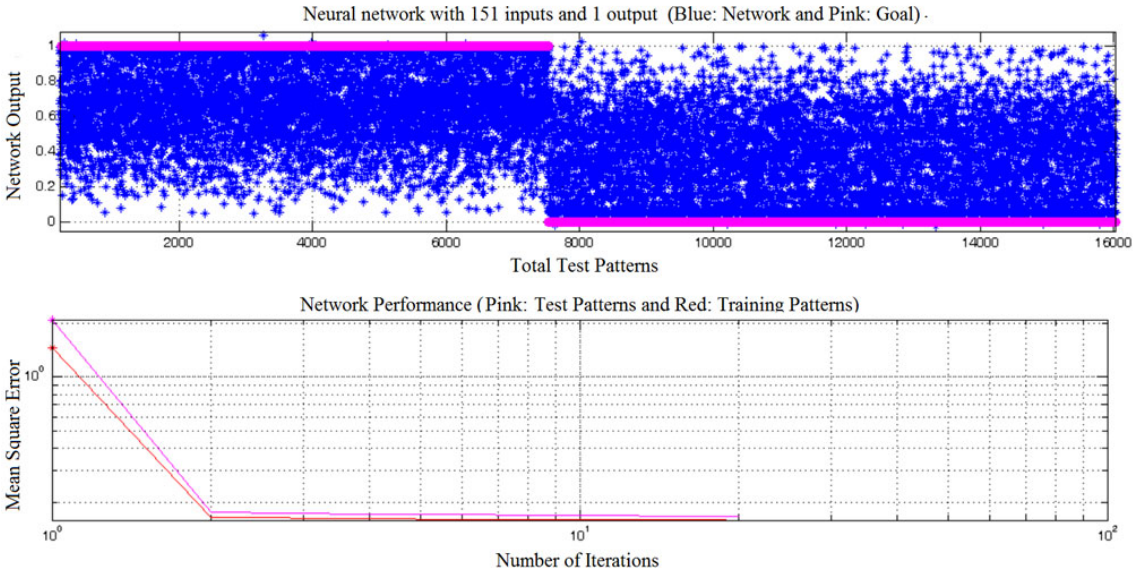

Figure 27. The Neural Training with balanced data

An important piece of information concerning Figure 27 is that the balanced data were obtained from non-balanced data, because as it is known, in a universe of credit card transactions, the total number of non-frauds is meaningfully dominant over the total number of frauds. The obtainment of balanced patterns is simple enough. In the case of Figure 27, frauds were multiplied by a factor of 15. It means that 37 thousand fraud transactions were changed into about 555 thousand, that is, almost the same number of non-fraud transactions (about 613 thousand). Evidently, this artifice causes the repetition of the same fraud several times.

In order to end this section, Figures 26 and 19 are compared. In Figure 26 (real world training data), the data in minority with output values that should be equal to one are centered in zero, which is sharply a non-Gaussian distribution, because in a Gaussian distribution, the data in majority should be centered in 1. In Figure 19 (theoretical simulation), a Gaussian distribution was considered and the data are really centered in 1, which is exactly the opposite of Figure 26. In fact, there is a difference between Monte Carlo theoretical model presented in this paper and the data of credit card frauds problem drawn from real world.

However, notice that the range of variation or dispersion of errors of data in minority (fraud) between the theoretical Monte Carlo method (see Figure 19) and the real world data (see Figure 26) is the same, and for being a first approximation, Monte Carlo simulation really got reasonable.

6. Conclusion

One can attempt to use Monte Carlo simulation to analyze and estimate the maximum acceptable mean square error of training and test patterns of a supervised learning, which uses some architecture of feedforward neural networks. The Monte Carlo method application also allows one to verify how the mean square error of test patterns varies according to the balance and unbalance of training patterns according to the total number of training patterns.

At first sight, it seems that balanced data can be trained with greater mean square errors, when compared to unbalanced patterns. So, one can list the following advantages of the Monte Carlo method application in the context of this paper:

(1) Knowing in advance what the maximum acceptable mean square error of neural training is;
(2) Give an a priori estimate of what neural network has a more favorable mean square error either networks with balanced either non-balanced data;
(3) Monte Carlo Simulation is almost instantaneous. It can offer an estimate of the training behavior of an MLP network at a mouse click. If one directly uses neural networks for that, it would take days or weeks and even then all simulation possibilities would not be embraced;
(4) Direct the neural training into a more acceptable direction and easier to be trained by the computer;
(5) Proving that balanced data can have greater training mean square error and test than non-balanced data and, therefore, it is easier to train artificial neural networks with balanced data.

Acknowledgements

The authors of this paper would like to thank the excellent technical reviews and written suggestions made by reviewers of this paper. They would also like to thank the support and infrastructure provided by the Brazilian Aeronautics Institute of Technology (Instituto Tecnológico de Aeronáutica - ITA) and the Casimiro Montenegro Filho Foundation (FCMF), and also the 2RP Net, a Brazilian Softwarehouse, for sponsoring this research.

References

[1] Metropolis N, Ulam S. The Monte Carlo method. Journal of the American statistical association, Taylor & Francis Broup. 1949; 44(247): 335-7. https://doi.org/10.1080/01621459.1949.10483310
[2] Hammersley JM. Monte Carlo methods. Methiren, London, 1964.
[3] Shreider YA. The Monte Carlo method. Pergamon Press, Oxford, 1967.
[4] Glasserman P. Monte Carlo methods in financial engineering: stochastic modeling and applied probability. p. 53. 1 ed., New York: Springer, 2004.
[5] Kroese DP, Taimre T, Botev ZI. Handbook of Monte Carlo methods. Wiley Series in probability and statistics, John Wiley & Sons, New York, 2011.
[6] Papoulis A. Probability, random variables, and stochastic processes. New York: McGraw-Hill, 1965.
[7] Jazwinski AH. Stochastic processes and filtering theory. New York and London: Academic Press, 1970.
[8] Vuolo JH. Fundamentos da teoria de erros. 2nd ed., São Paulo: Edgard Blücher Ltda, 1996.
[9] Cotter NE. The Stone-Weierstrass and its application to neural networks. IEEE Transactions on Neural Networks. 1990; 1(4): 290-6. PMid:18282849. https://doi.org/10.1109/72.80265
[10] Cybenko G. Continuous valued networks with two hidden layers are sufficient. Technical Report: Department of Computer Science, Tufts University, 1988.
[11] Haykin S. Neural networks: a comprehensive foundation. 2nd ed., New Jersey: Prentice-Hall, Inc., 1999.

[12] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Networks. 1989; 2(5): 359-8. https://doi.org/10.1016/0893-6080(89)90020-8

[13] Jang JSR, Sun CT, Mizutani E. Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence, Prentice-Hall, Inc., 1997.

[14] Tasinaffo PM, Guimarães RS, Dias LAV, et al. Discrete and exact general solution for nonlinear autonomous ordinary differential equations. International Journal of Innovative Computing, Information and Control. 2016; 12(5): 1703-17. https://doi.org/10.24507/ijicic.12.05.1703

[15] Figueiredo MO, Tasinaffo PM, Dias LAV. Modeling autonomous non linear dynamic systems using mean derivatives, fuzzy logic and genetic algorithms. International Journal of Innovative Computing, Information and Control. 2016; 12(5): 1721-23. https://doi.org/10.24507/ijicic.12.05.1721

[16] Russell S, Norvig P. Artificial intelligence. 2nd ed., New Jersey: Prentice-Hall, Inc., 2003.

[17] Spooner JT, Maggiore M, Ordónez R, et al. Stable adaptive control and estimation for nonlinear systems neural and fuzzy approximator techniques. New York: Wiley-Interscience, 2002.

[18] Zurada JM. Introduction to artificial neural system. St. Paul, MN, USA: West Pub. Co., 1992.

[19] Carneiro EM, Dias LAV, Cunha AM, et al. Cluster analysis and artificial neural networks: A case study in credit card fraud detection. In: 12th International Conference on Information Technology - New Generations, 2015, Las Vegas. Proceedings of the 12th International Conference on Information Technology. Los Alamitos: IEEE. p.122-126.

[20] Silvestre MR, Ling LL. Pruning methods to MLP neural networks considering proportional apparent error rate for classification problems with unbalanced data. Measurement. 2014; 56: 88-94. https://doi.org/10.1016/j.measurement.2014.06.018

[21] Wang Y, Li X, Ding X. Probabilistic framework of visual anomaly detection for unbalanced data. Neurocomputing. 2016; 201: 12-18. https://doi.org/10.1016/j.neucom.2016.03.038

[22] Lee CY, Lee ZJ. A novel algorithm applied to classify unbalanced data. Applied Soft Computing. 2012; 12(08): 2481-2485. https://doi.org/10.1016/j.asoc.2012.03.051

[23] Vajda S, Fink GA. Strategies for training robust neural network based digit recognizers on unbalanced data sets. In: 12th International Conference on Frontiers in Handwriting Recognition (ICFHR), 2010, Kolkata, India. IEEE Xplore, p. 148-153. https://doi.org/10.1109/ICFHR.2010.30