

Determining the optimal number of folds to use in a K-fold cross-validation: A neural network classification experiment

Opeoluwa Oyedele |

To cite this article: Opeoluwa Oyedele | (2023) Determining the optimal number of folds to use in a K-fold cross-validation: A neural network classification experiment, Research in Mathematics, 10:1, 2201015, DOI: [10.1080/27684830.2023.2201015](https://doi.org/10.1080/27684830.2023.2201015)

To link to this article: <https://doi.org/10.1080/27684830.2023.2201015>



© 2023 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.



Published online: 01 May 2023.



Submit your article to this journal [↗](#)



Article views: 1368



View related articles [↗](#)



View Crossmark data [↗](#)



Citing articles: 1 View citing articles [↗](#)

Determining the optimal number of folds to use in a K-fold cross-validation: A neural network classification experiment

Opeoluwa Oyedele 

Department of Computing, Mathematical and Statistical Sciences, School of Science, University of Namibia, Windhoek, Namibia

ABSTRACT

A large dataset is needed to obtain a large learning set for a suitable classifier, while a large testing set is needed for a good estimate of the classifier's performance (i.e. error probability). With a small dataset, after its random partitioning into learning and testing sets, both sets would end up consisting of smaller samples, which then becomes difficult to use when seeking to obtain a suitable classifier from the learning set and a good estimate of its performance from the testing set. The K-fold cross-validation approach has been every so often suggested to overcome the problem of not being able to obtain a suitable classifier and a good estimate of its performance. Thus, the objective of this study experiment was to determine the optimal number of folds to use in a K-fold cross-validation, and this was done in a simulation way using an artificial two-class normal mixture dataset with a total of 1000 samples and the resilient back propagation learning method over 10,000 training epochs, with and without early stopping applications during the training of the neural networks.

ARTICLE HISTORY

Received 15 June 2022
Accepted 05 April 2023

KEYWORDS



back propagation; cross-validation; error probability; network performance; neural networks; pattern classification

1. Introduction

Pattern classification can be considered as a technique/algorithm that can classify patterns in a dataset, with the aim of (automatically) classifying an object into one class ω_i of a set of c given classes $\{\omega_1, \omega_2, \dots, \omega_c\}$ (Webb, 2002). In this technique, several features x_1, x_2, \dots, x_n are extracted from the objects in a process called feature extraction and may be given by real numbers $x \in \mathbb{R}$ or in categories resulting from measurements of certain properties of the object, with their collection forming a feature vector $x = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ and often called a sample, in contrast to the usage of this concept in statistics. The collection of the feature vectors is referred to as a dataset, and once a feature vector is measured for an object, the classifier assigns it to one of the classes. In practice, classifiers are usually constructed directly from the dataset. To be precise, the dataset D is randomly divided into two parts, namely the learning set L and the testing set T , and in order to achieve independence of the two sets, the learning set is randomly selected from D (sometimes maintaining class proportions), while the leftover is taken as the testing set. The first part of D is called the learning set because the classifier is trained on it. Usually, the construction of a classifier from a given dataset involves the minimization of the number of

misclassifications in the learning set (Ripley, 2008; Webb, 2002). Thus, if the same learning set is used to estimate the performance (i.e. error probability) of the classifier, this estimate will be too optimistic. On the other hand, the testing set $T = D \setminus L$ is used to estimate the performance of the classifier as well as to construct confidence intervals for the true performance in order to obtain the accuracy level of the classification.

K-fold cross-validation is one of the methods most often suggested by researchers to overcome the problem of not being able to obtain a suitable classifier and a good estimate of its performance when dealing with small datasets. This is because a large dataset is needed to obtain a large learning set (in order to obtain a suitable classifier) and a large testing set (in order to obtain a good estimate of the classifier's performance). However, with a small dataset, after its random partition into learning and testing sets, both sets would end up consisting of smaller samples, which then becomes difficult to use when finding a suitable classifier from the learning set and a good estimate of its performance from the testing set. In K-fold cross-validation, the whole dataset D is randomly divided into K (equally sized) subsets/folds D_1, D_2, \dots, D_K such that the union of all these folds is equal to D and their pairwise intersection is an empty set. For each fold, a separate classifier is

CONTACT Opeoluwa Oyedele  OpeoluwaOyedele@gmail.com  Department of Computing, Mathematical and Statistical Sciences, School of Science, University of Namibia, Windhoek, Namibia
Hemen Dutta Gauhati University, INDIA

© 2023 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

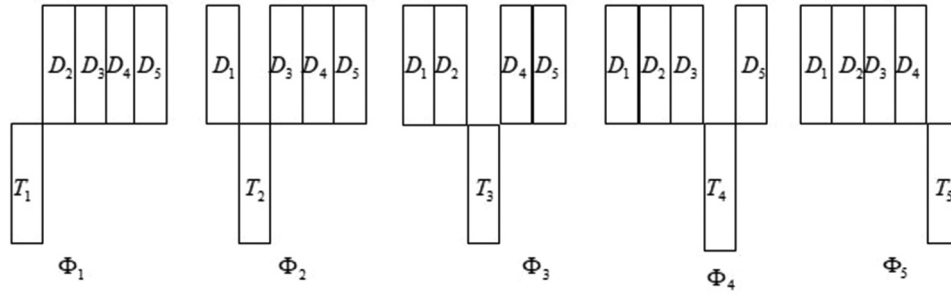


Figure 1. A schematic of a 5-fold cross-validation.

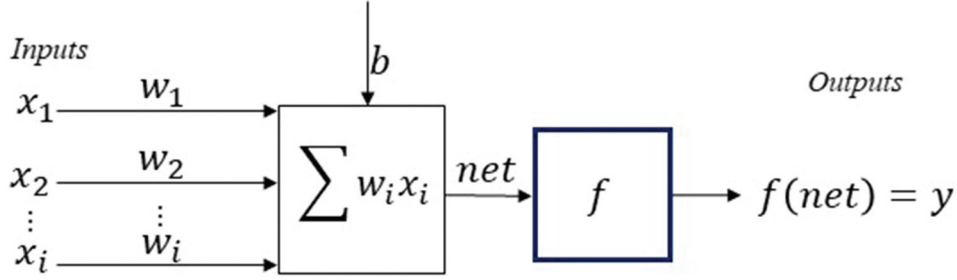


Figure 2. A schematic of a single neuron in neural network.

trained on $L_k = D \setminus D_k$ such that in the end, K different classifiers $\Phi_1, \Phi_2, \dots, \Phi_K$ are created from each of the folds. Furthermore, during the training of each of the classifiers, one of the folds will be left out to serve as the testing set for that classifier, as shown in Figure 1 for a 5-fold cross-validation scenario. The upper part of each Φ_k , for $k = 1, 2, \dots, 5$, in Figure 1 is used for training the respective classifier, while the lower part is used as the testing set for estimating the performance of that classifier. The general idea behind K -fold cross validation is to train K classifiers ($\Phi_1, \Phi_2, \dots, \Phi_K$), with each classifier trained using its own learning set $L_k = D \setminus D_k$ and its performance estimated using its own testing set $T_k = D_k$ respectively. Here, the basic assumption is that if Φ is a classifier trained on the whole dataset D and Φ_k are the classifiers trained on the learning set $L_k = D \setminus D_k$, then

$$P(\text{error}; \Phi) \approx P(\text{error}; \Phi_k) \quad (1)$$

for all $k = 1, 2, \dots, K$, and for large value of K (Kohavi, 1995). If K is the same as the size of D , then the K -fold cross-validation approach will be a leave-one-out method.

2. Neural network

Much of the research works done in neural networks have been inspired and influenced by knowledge of biological systems because they are based on the idea

of imitating the human brain (Rojas, 1996), with the neuron serving as the (basic) computing element in the systems (Patterson, 1998). Artificial neural networks are simplified models of the nervous system and are networks of highly interconnected neural computing elements that have the ability to respond to input stimuli and learn to adapt to the environments, with these elements operating in parallel (Patterson, 1998). Each input link has an associated external input signal (or stimulus) x_i and a corresponding weight w_i . The neuron behaves as an activation/mapping function $f(\cdot)$ producing an output $y = f(\text{net})$ as shown in Figure 2, where net is the weighted sum of the inputs stimuli to the neuron with a constant bias output b of $x_0 = +1$ and an adjustable weight $w_0 = b$ often included in its definition (Beale et al., 2017). To be precise,

$$\begin{aligned} \text{net} &= w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_ix_i \\ &= b + \sum w_ix_i \end{aligned}$$

and f (also referred to as a transfer function) is typically a non-linear monotonic non-decreasing function, such as a step function or a sigmoid function. The behaviour of a neural network depends on weights, biases and the input-output (transfer) function that is specified for the units (Patterson, 1998).

In general, training a network requires that the activation/mapping function be bounded and differentiable. One of the most commonly used functions satisfying these requirements is the sigmoid function (defined

over \mathbb{R} and bounded between 0 and 1). It is an S-shaped monotonic increasing function that has the general form of

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

where β is a constant that determines the steepness of the S-shaped curve. Other commonly used functions include the hyperbolic tangent function, log functions, trigonometric (sine and cosine) functions and radial basis (Gaussian) functions. Usually, there are three main layers (i.e. groups of neurons) in a network, namely the input layer, middle (hidden) layer and output layer. The input layer consists of neurons (nodes) receiving inputs directly from the outer part of the network, while the output layer consists of neurons whose outputs are passed to the world outside the network. The hidden layer consists of neurons that both receive internal inputs and produce internal outputs. However, these neurons are not connected to the network's (external) output as they are hidden from the outside world.

2.1. Neural network training

A neural network can be trained to perform a particular function by adjusting the values of the connections (weights) between elements so that a particular input leads to a specific target output while keeping the architecture (i.e. number of layers, number of neurons per layer and transfer function) fixed (Patterson, 1998). There are generally four rules to follow when training a network: (i) randomly divide the whole dataset into two parts and then reserve the first part as the learning set while the second part is taken as the testing set, (ii) create the network object, (iii) train the network by adapting parameters on the learning set, and (iv) simulate the network response to the testing set in order to estimate its performance.

Most often, the resilient back propagation learning method is used in training the network in which

corrections are made to the weights. The term “back propagation” refers to the manner in which the gradient of the performance/error function is computed for a non-linear multi-layer network and arises from the method in which corrections are made to the weights. The performance function determines how well the network is doing its task and can be defined using different measures such as the mean squared error measure or absolute error measure. Consider a learning set L with feature vectors $x^i \in \mathbb{R}^n$ and targets $t^i \in \mathbb{R}^n$. The performance function can be estimated as

$$E = \frac{1}{2} \sum_{i=1}^n \|\Phi(x^i) - t^i\|^2$$

where Φ is the input-output mapping function of the neural network (Patterson, 1998). During the training phrase, inputs patterns are presented to the network in sequences with each training pattern propagated forwardly, layer by layer, until an output pattern is computed. The computed output is then compared to a desired or target output, and an error value is determined. This error is later used as input to adjust the weights, layer by layer, in a backward direction.

Consider the neuron k at the output layer of a multi-layer network shown in Figure 3 and the situation at the hidden neuron j shown in Figure 4 for

$$z_k = g(\text{net}_k) = g(v_{k1}y_1 + v_{k2}y_2 + \cdots + v_{kP}y_P)$$

and

$$y_j = f(\text{net}_j) = f(w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jN}x_N).$$

The rules for updating the weights of the output layer and hidden layer units are as follows:

$$v_{ki} \leftarrow v_{ki} - \Delta v_{ki} \quad (2)$$

$$w_{ji} \leftarrow w_{ji} - \Delta w_{ji} \quad (3)$$

where

$$\Delta v_{ki} = \eta \delta_k^{\text{out}} y_i$$

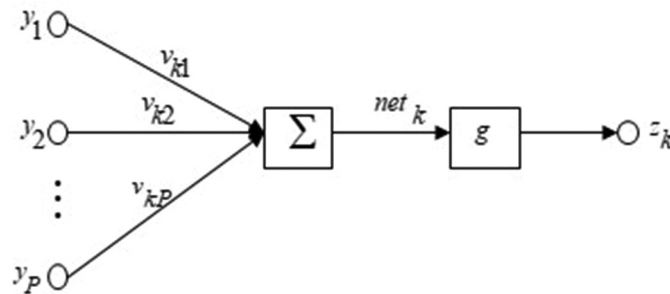


Figure 3. A schematic of neuron k at the output layer of a multi-layer network.

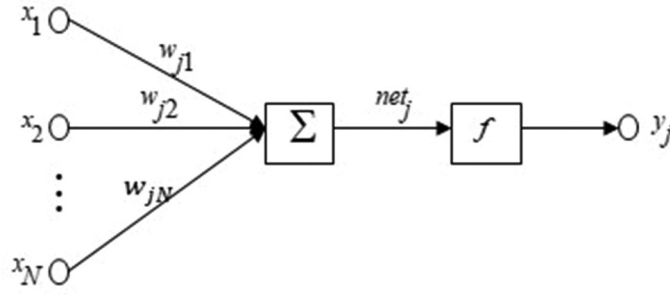


Figure 4. A schematic of neuron j at the hidden layer of a multi-layer network

$$\delta_k^{out} = (z_k - t_k) \cdot g'(net_k)$$

$$y^0 = x$$

$$\Delta w_{ji} = \eta \delta_j^{hidden} x_i$$

$$y^j = f_j(w^j y^{j-1})$$

$$\delta_j^{hidden} = \sum_{l=1}^M \delta_l^{out} v_{lj} f'(net_j)$$

Step 2 (Backward Passing): Using the values y^j computed by the final layer units and the corresponding target values t_j , compute

$$\delta_k^Q = (y_k^Q - t_k) \cdot f'_Q(net_k^Q)$$

afterwards, by back propagating the errors for each of the preceding layers, compute

$$\delta_k^j = \sum_l^M \delta_l^{j+1} w_{lk}^{j+1} \cdot f'_j(net_k^j)$$

for $j = Q - 1, Q - 2, \dots, 2, 1$.

Step 3 (Updating): Update the weights in layer j by

$$w_k^{j(new)} = w_k^{j(old)} - \eta \delta_k^j y^{j-1}$$

with $w_k = (w_{k1}, w_{k2}, \dots, w_{kN})$.

Repeat until the error over all learning samples is small enough.

for $\eta > 0$ constant learning coefficient/rate and t_k a target for neuron k (Riedmiller & Braun, 1993). In the resilient back propagation learning method, the hidden layer weights are adjusted using the errors from the subsequent layer. Thus, the errors computed at the output layer are used to adjust the weights between the last hidden layer and the output layer. Likewise, an error value computed from the last hidden layer output is used to adjust the weights in the next to the last hidden layer and so on until the weight connections to the first hidden layer are adjusted (Riedmiller, 1994). In this way, errors are propagated backwards, layer by layer, with corrections being made to the corresponding layer weights using equations (2) and (3) in an iterative manner. The process is repeated a number of times for each pattern in the learning set until the total output error converges to a minimum or until the maximum or some specified limit is reached in the number of learning iterations/epochs specified (Haykin, 1998; Patterson, 1998).

2.2. Resilient back propagation algorithm

Consider a multi-layer network with Q number of layers, where each layer is numbered from 1 to Q . The resilient back propagation algorithm can be summarized as follows.

Step 0 (Initialization): Initialize all weights w (and biases b) by small random numbers $\in [-\epsilon, \epsilon]$.

For each learning sample x ,

Step 1 (Forward Passing): Propagate this unit through the layers and compute the output, with inputs to layer 1 indexed with the superscript 0. That is, for $j = 1, 2, \dots, Q$

2.3. Overfitting and early stopping

Usually, fully connected multi-layer networks always have many parameters and are very prone to overfitting. Thus, when training a neural network, one is usually interested in obtaining a network with optimal generalization performance, i.e., small error on the unseen dataset (Geman et al., 1992). If a network is very complex and trained too much, the error may be driven to a very small value and during the testing phase, when new data are presented to the network, the error may become much larger than the error in the training process. When this happens, it means that the network fits the learning dataset so closely that it has not learned to generalize to new dataset sampled from the same distribution (Bishop, 2005; Duda et al., 2022). There

are two common ways of avoiding overfitting when training neural networks, especially the multi-layer ones. These are the simple network structure (i.e. non-early stopping) and the early stopping methods. A simple network structure or non-early stopping method involves reducing the number of dimensions of the parameter space (i.e. reducing the number of neurons and/or number of layers of neurons), while the early stopping method is used for achieving better performance on unseen data or improving generalization performance (Finnoff et al., 1993).

Early stopping method can be used either interactively (i.e. based on human judgement) or automatically (i.e. based on some formal stopping criterion imposed) and in its process the learning set is further split into two subsets, namely, the new learning set and the validation set. Here, the new-learning set is used for computing the gradient of the performance function and updating the network weights and biases. After each sweep through the new-learning set, the network is then evaluated on the validation set. On the other hand, the validation set is used in learning when to stop training the network and it prevents the over-adaptation of the learning set. In addition, the error on the validation set is monitored during the training process and normally decreases during the initial phase of training, so does the new-learning set error. However, when the network begins to overfit the data, the error in the validation set begins to rise. When this happens for a specified number of training iterations, the training is immediately stopped, and the weights and biases at the minimum of the validation error are returned. Afterwards, the network with the best performance on the validation set is later used for the actual testing and classification. It should be noted that the testing set is not used at all during the training process of the network, but it is only used to verify the network design (i.e. estimate the performance of the network).

In general, the early stopping method involves five basic steps: (i) divide the whole dataset randomly into learning set and testing set, (ii) randomly split the

learning set into new-learning set and validation set, (iii) use a large number of hidden units, (iv) compute the validation error rate periodically during training, and (v) stop the training when the validation error rate starts going up. There are several advantages of using the early stopping method such as: (i) computational speed, (ii) its applicable to networks in which the number of weights far exceeds the sample size, and (iii) only two major decisions are required by the user—what proportion of the validation cases to use and how many passes through the learning set is needed in order to see if the error rate goes up (Finnoff et al., 1993).

3. Methodology

The MATLAB programming language version 9.2 (R2017a) with the Neural Network Toolbox version 10.0 (R2017a) software was used in carrying out the neural network classification experiment of this study on a 64-bit operating system computer fitted with Windows 10 Pro 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40 GHz and 8.00GB of RAM. The idea of this experiment was to estimate the error probabilities with K-fold cross-validation on an artificial two-class normal mixture dataset and afterwards compare them to the estimated error rate obtained from a bigger (independent) testing set sampled from the same distribution as the artificial dataset in order to determine the optimal number of folds to use in a K-fold cross-validation.

3.1. Data

The dataset used in this experiment was an artificial two-class normal mixture dataset, with the first class (ω_1) having three sub-classes and the second class (ω_2) having two sub-classes, all randomly generated in MATLAB using the parameters shown in Table 1. Additionally, the whole dataset had a total of 1000 samples, out of which 500 was assigned to class ω_1 and the remaining 500 assigned to class ω_2 . For class ω_1

Table 1. Parameters for the two-class normal mixture data

Class	Proportions	Mean Vectors	Covariance Matrix	Size
ω_1	$\pi_1 = 0.4$ $\pi_2 = 0.3$ $\pi_3 = 0.3$	$\mu_1 = \begin{pmatrix} 6 \\ 7 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 2.5 & 0.9 \\ 0.9 & 1.4 \end{pmatrix}$	500
		$\mu_2 = \begin{pmatrix} 7 \\ 7 \end{pmatrix}$	$\Sigma_2 = \begin{pmatrix} 0.3 & 0 \\ 0 & 0.3 \end{pmatrix}$	
		$\mu_3 = \begin{pmatrix} 4 \\ 9 \end{pmatrix}$	$\Sigma_3 = \begin{pmatrix} 1.8 & -0.2 \\ -0.2 & 0.4 \end{pmatrix}$	
ω_2	$\pi_1 = 0.4$ $\pi_2 = 0.6$	$\mu_1 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$	$\Sigma_1 = \begin{pmatrix} 1 & -0.7 \\ -0.7 & 2.3 \end{pmatrix}$	500
		$\mu_2 = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$	$\Sigma_2 = \begin{pmatrix} 2 & 0.8 \\ 0.8 & 1.7 \end{pmatrix}$	

having three sub-classes, 40% of its samples were from the first sub-class, while 30% of its samples were from the second sub-class and the remaining 30% from the third sub-class. Likewise, for class ω_2 having two sub-classes, 40% of its samples were from the first sub-class, while the remaining 60% were from the second sub-class as shown in Table 1. In this study, the target outputs used were randomly generated in MATLAB and were binary coded with 0 for belonging to the first class (ω_1) and 1 for belonging to the second class (ω_2).

3.2. Neural network section

3.2.1. Learning, testing and validation sets

In this study, random partitioning of the whole dataset into learning and testing sets was done for each number of folds before training the network. However, in the early stopping method, the learning set (for each number of folds) was further partitioned (randomly) with 25% of its data used as validation set and the rest as the new-learning set. Moreover, a (bigger) independent testing set of size 10,000 (independently sampled from the same distribution and using the same parameters in Table 1 but with 5000 samples per class) was created in order to estimate the performance (i.e. error probability) of the network.

3.2.2. Cross-validation

In this (neural network) classification experiment, various numbers of folds were used and they ranged from 2 to $N = 1000$ (where N was the total number of samples in the whole dataset). In addition, 999 classifiers $\Phi_2, \Phi_3, \dots, \Phi_{1000}$ were created and trained. The resilient back propagation learning method was used in training the neural networks with a maximum value of 10,000 training epochs and a sigmoid transfer function. In order to determine the optimal number of hidden neurons to be used in the network, random partitioning of the whole dataset was done again into learning and testing sets, of which 50% of the data was used as the learning set, while the remaining 50% was used as the testing set.

3.3. Performance (or error probability)

Most of the following derivations below were derived from information presented in Breiman and Spector (1992), Kohavi (1995), Webb (2002), Bishop (2005), Ripley (2008), Breiman et al. (2017) and Duda et al. (2022).

3.3.1. Unbiased estimator for the error probability

To obtain an unbiased estimate of the error probability $\hat{P}(\text{error})$ of the classifier Φ trained on the whole dataset D based on the basic assumption of cross-validation (Equation (1)), let

$N_{jj}^{(k)}$ = Number of class ω_j samples in $T_k = D_k$ classified as ω_j by Φ_k
 $N_{jj} = \sum_{k=1}^K N_{jj}^{(k)}$ = Total number of class ω_j samples in D classified as ω_j
 $N_j^{(k)} = \frac{N}{K}$ = Number of samples in T_k
 $N_j^{(k)}$ = Number of class ω_j samples in $T_k = D_k$
 $N_j = \sum_{k=1}^K N_j^{(k)}$ = Number of class ω_j samples in D
 $P(\omega_j)$ = Prior probability of class ω_j samples
 for $j = 1, 2, \dots$, and $k = 1, 2, \dots, K$ folds.

Furthermore, consider a random variable Z for classifier Φ trained on D as

$$Z(x, t) = \begin{cases} 1 & \text{if } \Phi(x) \neq t \\ 0 & \text{else} \end{cases}$$

for a feature vector x and target t , with $E(Z) = P(\Phi(x) \neq t)$ and

$$E(Z|\omega_j) = P(\Phi(x) \neq t|\omega_j) = P(\text{error}|\omega_j).$$

The error probability can be defined as

$$P(\text{error}) = \sum_{j=1}^c P(\text{error}; \omega_j) = \sum_{j=1}^c P(\omega_j) E(Z|\omega_j).$$

Since the unbiased estimator for $E(Z|\omega_j)$ is the sample mean of class ω_j samples, i.e.

$$\frac{1}{N_j} \sum_{\substack{(x, t) \in D, \\ t = \omega_j}} Z(x, t) = 1 - \frac{N_{jj}}{N_j}$$

the unbiased estimator for $P(\text{error})$ will be

$$\hat{P}(\text{error}) = \sum_{j=1}^c P(\omega_j) \left(1 - \frac{N_{jj}}{N_j}\right).$$

If the priors are reflected as the proportions of the classes in the testing set, i.e. $P(\omega_j) = \frac{N_j}{N}$ for all classes $\{\omega_1, \omega_2, \dots, \omega_c\}$, then $\hat{P}(\text{error})$ becomes

$$\hat{P}(\text{error}) = 1 - \frac{1}{N} \sum_{j=1}^c N_{jj}$$

with $\sum_{j=1}^c P(\omega_j) = 1$. Moreover, since the proportion of class ω_j samples in the testing set is approximately equal to the proportion of class ω_j samples in D ,

$$\frac{N_j^{(k)}}{N^{(k)}} \approx \frac{N_j}{N} \Rightarrow N_j \approx \frac{NN_j^{(k)}}{N^{(k)}}$$

and

$$N^{(k)} \approx \frac{N}{K} \Rightarrow N = N^{(k)}K$$

making $\frac{1}{N_j} = \frac{1}{KN_j^{(k)}}$. Thus, $\hat{P}(\text{error})$ becomes

$$\begin{aligned} \hat{P}(\text{error}) &= \sum_{j=1}^c P(\omega_j) \left(1 - \frac{N_{jj}}{N_j}\right) \\ &= \sum_{j=1}^c P(\omega_j) \frac{1}{N_j} \sum_{(x,t) \in D} Z((x, t)) \\ &= \sum_{j=1}^c P(\omega_j) \sum_{k=1}^K \frac{1}{KN_j^{(k)}} \sum_{(x,t) \in D_k} Z((x, t)) \\ &= \frac{1}{K} \sum_{k=1}^K \sum_{j=1}^c P(\omega_j) \frac{1}{N_j^{(k)}} \left(N_j^{(k)} - N_{jj}^{(k)}\right) \\ &\quad \hat{P}(\text{error}; \Phi_k) \\ &= \frac{1}{K} \sum_{k=1}^K \hat{P}(\text{error}; \Phi_k). \end{aligned} \quad (4)$$

From equation (4), it can be deduced that

$$\hat{P}(\text{error}; \Phi_k) = \sum_{j=1}^c P(\omega_j) \frac{1}{N_j^{(k)}} \left(N_j^{(k)} - N_{jj}^{(k)}\right)$$

and

$$\hat{P}(\text{error}; \Phi) = \frac{1}{K} \sum_{k=1}^K \hat{P}(\text{error}; \Phi_k).$$

In other words, the estimated error probability of classifier Φ trained on D is the average of the estimated error probabilities of classifiers Φ_k trained on the learning set $L_k = D \setminus D_k$ estimated on D_k .

3.3.2. Unbiased estimator for the standard error

To obtain the standard error (SE) of the error probability (equation (4)), let $\hat{P} = \hat{P}(\text{error})$, and

$$\begin{aligned} \hat{P}_j^{(k)} &= \sum_{k=1}^K \frac{1}{N_j^{(k)}} \left(N_j^{(k)} - N_{jj}^{(k)}\right) \\ \hat{P}_j &= \frac{1}{K} \sum_{k=1}^K \hat{P}_j^{(k)}. \end{aligned}$$

From equation (4),

$$\begin{aligned} \hat{P} &= \sum_{j=1}^c P(\omega_j) \hat{P}_j \\ &= \sum_{j=1}^c P(\omega_j) \frac{1}{N_j} \sum_{\substack{(x,t) \in D, \\ t = \omega_j}} Z(x, t). \end{aligned}$$

Since Z is binomially distributed with N_j number of trials and $P(\text{error}|\omega_j)$ is the probability of success, the variance of \hat{P} can be defined as

$$\text{Var}(\hat{P}) = \sum_{j=1}^c (P(\omega_j))^2 \frac{1}{N_j^2} N_j \text{Var}(Z|\omega_j)$$

where $\text{Var}(Z|\omega_j)$ can be estimated by the sample variance $s_j^2 = \hat{P}_j(1 - \hat{P}_j)$ of the sample $Z(x, t)$ with $t = \omega_j$.

Thus, the unbiased SE of $\hat{P}(\text{error})$ can be estimated as

$$\begin{aligned} \text{SE}(\hat{P}) &= \sqrt{\sum_{j=1}^c (P(\omega_j))^2 \frac{1}{N_j} \hat{P}_j(1 - \hat{P}_j)} \\ &= \sqrt{\sum_{j=1}^c (P(\omega_j))^2 \frac{s_j^2}{N_j}} \end{aligned} \quad (5)$$

where $\hat{P} = \hat{P}(\text{error})$.

3.3.3. Unbiased estimator for the confidence interval

The estimated SE of the error probability (equation (5)) can be used in the estimation of the confidence interval (CI) of the error probability. If the sample size N is large enough, by the central limit theorem,

$$\frac{\hat{P} - P}{\sigma_{\hat{P}}} \sim N(0, 1)$$

Given the confidence level $(1 - \alpha)$, the z -score can be determined such that

$$P_r \left(-z \leq \frac{\hat{P} - P}{\sigma_{\hat{P}}} \leq z \right) = 1 - \alpha$$

and solving for $P = P(\text{error})$ yields

$$P_r(\hat{P} - z\sigma_{\hat{P}} \leq P(\text{error}) \leq \hat{P} + z\sigma_{\hat{P}}) = 1 - \alpha$$

where α is the level of significance. Hence, with a probability of $(1 - \alpha)$, the true performance (or error probability) is within the interval $[\hat{P} \pm z\sigma_{\hat{P}}]$. Moreover, using the estimated SE obtained in equation (5), the unbiased CI for $\hat{P} = \hat{P}(\text{error})$ would become

$$[\hat{P} \pm z_{\alpha/2} \text{SE}(\hat{P})] \quad (6)$$

3.3.4. With and without early stopping method

Using a simple network structure to avoid overtraining the network, out of the 1 to 100 neurons considered, the lowest error rate was observed at the 6th neuron, as shown in Figure 5. Thus, in this study, six hidden neurons were used in the final training of the network without applying the early stopping method. However, since the number of hidden neurons is not of importance under the early stopping method and the training of the network only stops when the error rate on the

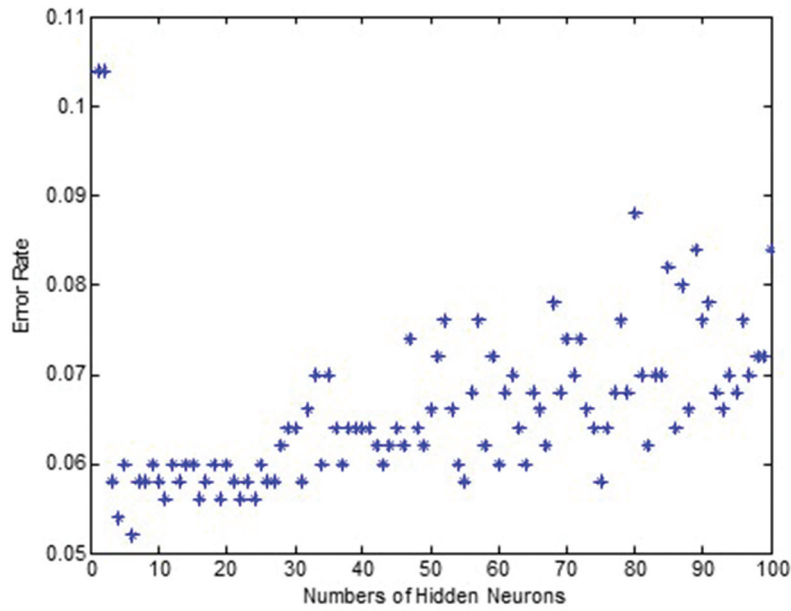


Figure 5. A plot of the number of hidden neurons and their respective error rate

validation set starts to increase, 100 hidden neurons were used in the training of the network, with the early stopping method.

4. Results & discussions

4.1. Without early stopping method

From the resilient back propagation trained network with six hidden neurons in its hidden layer, a sigmoid transfer function with 10,000 training epochs, and without the application of early stopping

method in its training, the training of the network for each number of folds stopped when the 10,000th training epochs was reached, while the error rate from the learning set kept going down even after the 10,000th iteration for each number of folds as shown in Figure 6. Using 1000 folds, in Figure 6, it can be observed that the error rate on the learning set kept going down but remained constant after the 4000th iteration until the 10,000th training epochs was reached, and if the number of iterations were to be increased, the error rate from this learning set might still remain constant.

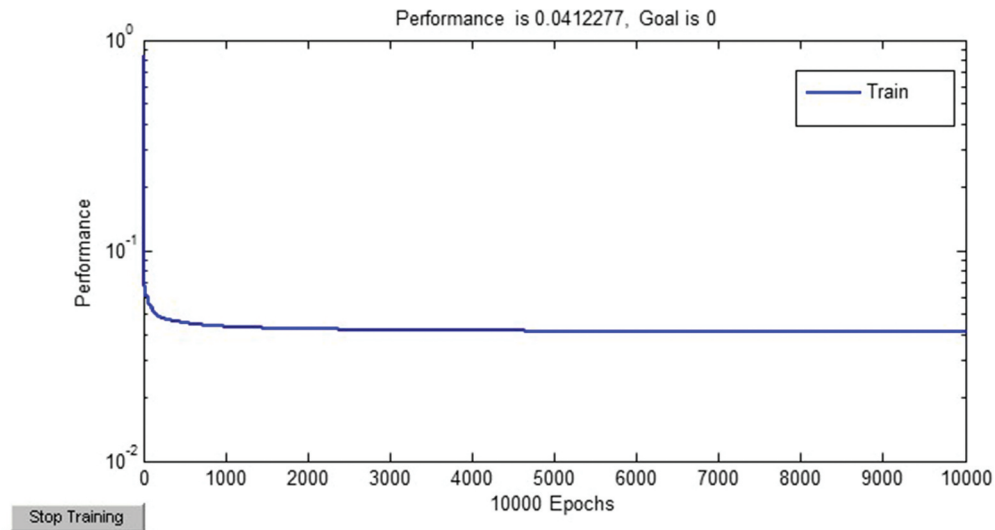


Figure 6. A plot of the network training error rates without early stopping for 1000 folds.

Table 2. Closest error rates without early stopping

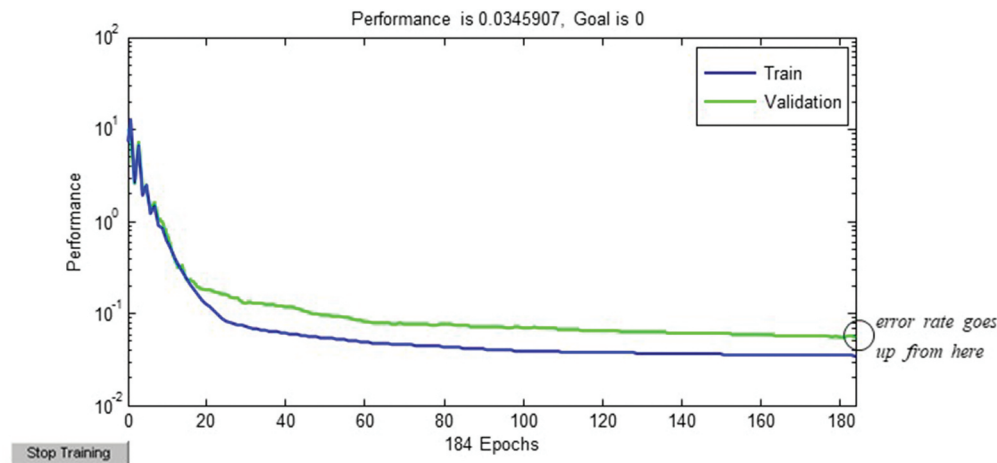
	Error rate	Standard error	90% confidence interval
Testing set	0.0503	0.00062	[0.04928, 0.05132]
K = 10 folds	0.0503	0.00014	[0.05007, 0.05053]
K = 15 folds	0.0503	0.00072	[0.04912, 0.05148]
K = 19 folds	0.0503	0.00078	[0.04902, 0.05158]
K = 16 folds	0.0502	0.00076	[0.04905, 0.05155]
K = 20 folds	0.0502	0.00082	[0.04885, 0.05155]

To determine the optimal number of folds, after the completion of the network training, the estimated error rate (equation (4)) per number of folds as well as their confidence intervals (equation (6)) were estimated and later on compared with the rate obtained from the independent testing set whose network was also trained without the application of the early stopping method. Detailed results of all the obtained estimated error rate (equation (4)) on each number of folds with their estimated standard error (equation (5)) and 90% confidence interval (equation (6)) without the application of the early stopping method can be obtained by running the MATLAB codes in the Appendix section of this article. An estimated error rate of 0.0503 (5.03%) was obtained from the independent testing set. By comparing this rate to the estimated error rate per number of folds, the closest rate occurred when $K = 10$, $K = 15$ and $K = 19$ folds were used in the cross-validation, followed by the ones obtained when $K = 16$ and $K = 20$ folds were used as shown in Table 2. Moreover, after estimating the standard error for each error rate, their confidence interval was estimated, and it was observed that some overlapping occurred between these intervals. Looking at the confidence intervals in Table 2, all five confidence intervals were overlapping with the interval obtained for the independent testing set. Furthermore, from this experiment, the error rate per number of folds kept fluctuating but had a minimum value of 0.0163

(1.63%) when $K = 521$ folds were used and a maximum value of 0.0617 (6.17%) when $K = 84$ folds were used, with their respective 90% confidence intervals not overlapping with that of the independent testing set. This is not surprising since the error rate of 0.0503 was not within both confidence intervals as the minimum error rate had an interval [0.01566, 0.01694], while the maximum error rate had an interval [0.06015, 0.06325]. Also, looking at the error rate per number of folds, it was observed that after $K = 244$ folds were used, the obtained error rate per number of folds kept drifting away (significantly) from the error rate obtained from the independent testing set (0.0503) as the number of folds increased. For these reasons, without early stopping application, it can be concluded that when the number of folds was large, it underestimated the true performance of the network, which implied that the leave-one-out method (when K is the same as the size of the whole dataset) was not good. Thus, using the optimal number, which is between 10 and 20 from this experiment, for K is prudent.

4.2. With early stopping method

From the resilient back propagation trained network with 100 hidden neurons in its hidden layer, a sigmoid transfer function with 10,000 training epochs, and with the application of early stopping method in its training, the training of the network was stopped earlier for each number of folds and not when the 10,000th training epochs was reached. This was due to the restriction of immediately stopping the network training when the error rate on the validation set starts going up. Using 1000 folds, in Figure 7, the error rate went up after the 184th iteration, which implied that the training of the

**Figure 7.** A plot of the network training error rates with early stopping for 1000 folds

network when 1000 folds were used was stopped after the 184th iteration. Also, the error rate from the learning set kept going down and seemed constant after the 184th iteration as observed in Figure 7.

To determine the optimal number of folds, after the completion of the network training which was stopped earlier, the estimated error rate (equation (4)) per number of folds as well as their confidence intervals (equation (6)) were determined and later on compared with the rate obtained from the independent testing set whose network was also trained with the application of the early stopping method. Detailed results of all the obtained estimated error rate (equation (4)) on each number of folds with their estimated standard error (equation (5)) and 90% confidence interval (equation (6)) with the application of the early stopping method can be obtained by running the MATLAB codes in the Appendix section of this article. An estimated error rate of 0.0807 (8.07%) was obtained from the independent testing set. By comparing this rate to the estimated error rate per number of folds, the closest rate occurred when $K = 20$ and $K = 13$ folds were used in the cross-validation, followed by the ones obtained when $K = 10$, $K = 15$ and $K = 19$ folds were used as shown in Table 3. Moreover, after estimating the standard error for each error rate, their confidence interval was estimated, and it was observed that some overlapping occurred between these intervals. Looking at the confidence intervals in Table 3, all five confidence intervals were strongly overlapping with the interval obtained for the independent testing set. Furthermore, from this experiment, the error rate per number of folds kept fluctuating but had a minimum value of 0.0392 (3.92%) when $K = 536$ folds were used and a maximum value of 0.245 (24.5%) when $K = 6$ folds were used, with their respective 90% confidence intervals not overlapping with that of the independent testing set. This is not startling since the error rate of 0.0807 was not in both intervals as the minimum error rate had an interval [0.03825, 0.04015], while the maximum error rate had an interval [0.24354, 0.24646]. Additionally, by looking at the error rate per number of folds, it can be seen that after $K = 494$ folds were used, the obtained error rate per number of folds kept drifting away (significantly) from the error rate obtained from the independent testing set (0.0807) as the number of folds kept increasing. For

these reasons, with early stopping application, it can be concluded that when the number of folds was large, it underestimated the true performance of the network, which implied that the leave-one-out method was not good. Hence, using the optimal number, which is between 10 and 20 from this experiment, for K is sensible.

4.3. Some remarks & limitations

In this classification experiment, the number of folds used in the K -fold cross-validation did affect the results such that as the number of folds were increasing, the obtained error rate per number of folds kept drifting away from the (independent) testing set's error rate, thereby underestimating the true performance of the classification. Additionally, to ensure that the selected K is providing the best generalization performance, its estimated error rate (equation (4)) must be the closest to the rate obtained from the independent testing set. Alternatively, a test of the predictive significance per number of folds can be determined using predictive test methods, such as the prediction error sum-of-squares measure and the root mean squared error of prediction measure, and this can be used to further determine the appropriate number of folds to use. Here, the K with the lowest prediction error value can then be chosen as the best number of folds to use for generalization performance. Moreover, although the K -fold cross-validation involves multiple iterations and testing thereby making it less prone to selection bias and giving a more accurate result, increasing the number of K can result in more time being spent on training, thereby making the training process more computationally expensive and time-consuming. Likewise, higher values of K underestimate the true performance of the classification and lead to a leave-one-out approach that is not good and even more computationally expensive.

5. Summary

With the idea of estimating performance/error probabilities with K -fold cross-validation and afterwards comparing them to the error rate obtained from the bigger independent testing set with 10,000 samples (randomly) sampled from the same distribution as the artificial dataset, it can be concluded that the optimal number of folds ranged from 10 to 20 whether early stopping method was applied in the network training or not. The (independent) testing set error rate of 5% obtained from the simple structure network without early stopping method being applied in its training was lower compared to the one obtained from the network when the early stopping method was applied in its training (8%). Thus, it is recommended that one could look into repeated experiments to

Table 3. Closest error rates with early stopping

	Error rate	Standard error	90% confidence interval
Testing set	0.0807	0.00104	[0.07899, 0.08241]
$K = 20$ folds	0.0803	0.00093	[0.07877, 0.08183]
$K = 13$ folds	0.0802	0.00082	[0.07885, 0.08155]
$K = 10$ folds	0.0801	0.00088	[0.07865, 0.08155]
$K = 15$ folds	0.0801	0.00080	[0.07878, 0.08142]
$K = 19$ folds	0.0801	0.00084	[0.07872, 0.08148]

see how stable these error values are. Moreover, as the number of folds increased, the obtained error rate per number of folds kept drifting away from the (independent) testing set's error rate, thereby underestimating the true performance of the classification.

Acknowledgements

Prof G. Jäger is thanked for his guidance.

Disclosure statement

No potential conflict of interest was reported by the author.

Funding

The author received no financial support or any specific grant from funding agencies in the public, commercial, or non-profit organisation sectors for the research, authorship, and/or publication of this article.

ORCID

Opeoluwa Oyedele  <http://orcid.org/0000-0002-3053-0702>

Data availability statement

The dataset used in this study was an artificial two-class normal mixture dataset generated using the MATLAB programming language version 9.2 (R2017a), with the MATLAB codes provided in the Appendix section of this article.

References

- Beale, M. H., Hagan, M. T., & Demuth, H. B. (2017). *Neural network toolbox user's guide*. The MathWorks Inc.

- Bishop, C. M. (2005). *Neural networks for pattern recognition*. Oxford University Press Inc.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (2017). *Classification and regression trees*. CRC Press.
- Breiman, L., & Spector, P. (1992). Submodel selection and evaluation in regression: The X-random case. *International Statistical Review/Revue Internationale de Statistique*, 60(3), 291–319. <https://doi.org/10.2307/1403680>
- Duda, R. O., Hart, P. E., & Stork, D. G. (2022). *Pattern classification* (3rd ed.). Wiley.
- Finnoff, W., Hergert, F., & Zimmermann, H. G. (1993). Improving model selection by nonconvergent methods. *Neural Networks*, 6(6), 771–783. [https://doi.org/10.1016/S0893-6080\(05\)80122-4](https://doi.org/10.1016/S0893-6080(05)80122-4)
- Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1), 1–58. <https://doi.org/10.1162/neco.1992.4.1.1>
- Haykin, S. (1998). *Neural networks: A comprehensive foundation*. Prentice Hall.
- Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the 14th international joint conference on artificial intelligence*, Montreal, Quebec, Canada, 2, 1137–1143.
- Patterson, D. W. (1998). *Artificial neural networks: Theory and applications*. Prentice Hall.
- Riedmiller, M. (1994). *Resilient back propagation - description and implementation details*. Technical Report, University of Karlsruhe.
- Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster back propagation learning: The RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Network*, San Francisco, CA, USA, 1, 586–591. IEEE.
- Ripley, B. D. (2008). *Pattern recognition and neural networks*. Cambridge University Press.
- Rojas, R. (1996). *Neural networks: A systematic introduction*. Springer.
- Webb, A. (2002). *Statistical pattern recognition* (2nd ed.). Wiley.

Appendix

The following MATLAB functions were used in the neural network classification experiment of this study.

```

N1 = 500;
N2 = 500;
N=N1+N2
mu1=[6,4];
mu2=[7,7];
mu3=[4,9];
mu4=[2,2];
mu5=[2,6];
sigma1=[2.59;9,1.4];
sigma2=[0.3,0;0,0.3];
sigma3=[1.8,-.2;-.24];
sigma4=[1,-.7;-.7,2.3];
sigma5=[28;8,1.7];
x1=mvnrnd(repmat(mu1,0.4*N1,1),sigma1);
x2=mvnrnd(repmat(mu2,0.3*N1,1),sigma2);
x3=mvnrnd(repmat(mu3,0.3*N1,1),sigma3);
x=[x1;x2;x3]; %random samples for class 1
y1=mvnrnd(repmat(mu4,0.4*N2,1),sigma4);
y2=mvnrnd(repmat(mu5,0.6*N2,1),sigma5);
y=[y1;y2]; %random samples for class 2
data=[x;y];
for K = 2:N
    index1=find(targets == 0);
    l1=length(index1);
    len=floor(l1/K);
    rand('state',sum(100*clock))
    randindex1=randperm(l1);
    index2=find(targets == 1);
    l2=length(index2);
    len2=floor(l2/K);
    randindex2=randperm(l2);
    for k = 1:K
        test_index1=index1(randindex1((k-1)*len+1:k*len));
        test_index2=index2(randindex2((k-1)*len2 + 1:
            k*len2));
        test_index=[test_index1;test_index2];
        test_data=data(test_index,:);
        train_index1=setdiff(index1,test_index1);
        train_index2=setdiff(index2,test_index2);
        train_index=[train_index1;train_index2];
        train_data=data(train_index,:);
    end
end
p = 0.25;
val_index1=train_index1(1:round(p*length(train_index1)));
val_index2=train_index2(1:round(p*length(train_index2)));
val_index=[val_index1;val_index2];
val_data=data(val_index,:);
new_train_index1=setdiff(train_index1,val_index1);
new_train_index2=setdiff(train_index2,val_index2);
new_train_index=[new_train_index1;new_train_index2];
new_train_data=data(new_train_index,:);
M1 = 5000;
M2 = 5000;
M=M1+M2
X1=mvnrnd(repmat(mu1,0.4*M1,1),sigma1);
X2=mvnrnd(repmat(mu2,0.3*M1,1),sigma2);
X3=mvnrnd(repmat(mu3,0.3*M1,1),sigma3);

```

```

X=[X1;X2;X3]; %random samples for class 1
Y1=mvnrnd(repmat(mu4,0.7*M2,1),sigma4);
Y2=mvnrnd(repmat(mu5,0.3*M2,1),sigma5);
Y=[Y1;Y2]; %random samples for class 2
data_test=[X;Y];

```

```

%without early stopping
targets=[ones(N1,1)*0;ones(N2,1)*1];
train_targets=targets(train_index);
test_targets=targets(test_index);
for K = 2:N
    for k = 1:K
        net=newff(minmax(data'),[6,1],
            {'logsig','purelin'},'trainrp');
        net.trainParam.epochs = 10000;
        net=train(net,train_data',train_targets');
        out=sim(net,test_data');
        class=round(out);
        check=length(find(class~=test_targets'));
        error_rate(k)=check/length(out);
    end
    total_error(K)=sum(error_rate)/K
    index11=find(test_targets == 0);
    index22=find(test_targets == 1);
    Class=class';
    s1=var(Class(index11)~=test_targets(index11));
    s2=var(Class(index22)~=test_targets(index22));
    SE(K)=(1/N)*sqrt(s1+s2)
end
big_test_targets=[ones(M1,1)*0;ones(M2,1)*1];
net_test=newff(minmax(data'),[6,1],
    {'logsig','purelin'},'trainrp');
net_test.trainParam.epochs = 10000;
net_test=train(net_test,data',targets');
out_test=sim(net_test,data_test');
class_test=round(out_test);
check_test=length(find(class_test~=big_test_targets'))
error_rate_test=check_test/M
index11_test=find(big_test_targets == 0);
index22_test=find(big_test_targets == 1);
Class_test=class_test';
s1_test=var(Class_test(index11_test)~=big_test_targets
    (index11_test));
s2_test=var(Class_test(index22_test)~=big_test_targets
    (index22_test));
SE_test=(1/M)*sqrt(s1_test+s2_test)
level = 0.90;
z=norminv(1-(1-level)/2)
lower_limit=total_error-(z*SE)
upper_limit=total_error+(z*SE)
lower_limit_test=error_rate_test-(z*SE_test)
upper_limit_test=error_rate_test+(z*SE_test)
w1=net.IW{1}
b1=net.b{1}
w2=net.LW{2,1}
b2=net.b{2}

```

```

%with early stopping
targets=[ones(N1,1)*0;ones(N2,1)*1];
val_targets=targets(val_index);
for K = 2:N
    for k = 1:K

```

```

net=newff(minmax(data'),[100,1],
{'logsig','purelin'},'trainrp');
net.trainParam.epochs = 10000;
net=init(net);
val.P=val_data';
val.T=val_targets';
new_net=train(net,new_train_data',new_train_targets',[],[],val);
out=sim(new_net,test_data');
class=round(out);
check=length(find(class~=test_targets'));
error_rate(k)=check/length(out);
end
total_error(K)=sum(error_rate)/K
index11=find(test_targets==0);
index22=find(test_targets==1);
Class=class';
s1=var(Class(index11)~=test_targets(index11));
s2=var(Class(index22)~=test_targets(index22));
SE(K)=(1/N)*sqrt(s1+s2)
end
net_test=newff(minmax(data'),[100,1],
{'logsig','purelin'},'trainrp');
net_test.trainParam.epochs = 10000;

```

```

net_test=train(net_test,data',targets',[],[],val);
out_test=sim(net_test,data_test');
class_test=round(out_test);
check_test=length(find(class_test~=big_test_targets'))
error_rate_test=check_test/M
index11_test=find(big_test_targets==0);
index22_test=find(big_test_targets==1);
Class_test=class_test';
s1_test=var(Class_test(index11_test)~=big_test_targets(index11_test));
s2_test=var(Class_test(index22_test)~=big_test_targets(index22_test));
SE_test=(1/M)*sqrt(s1_test+s2_test)
level = 0.90;
z=norminv(1-(1-level)/2)
lower_limit=total_error-(z*SE)
upper_limit=total_error+(z*SE)
lower_limit_test=error_rate_test-(z*SE_test)
upper_limit_test=error_rate_test+(z*SE_test)
W1=new_net.IW{1}
B1=new_net.b{1}
W2=new_net.LW{2,1}
B2=new_net.b{2}

```