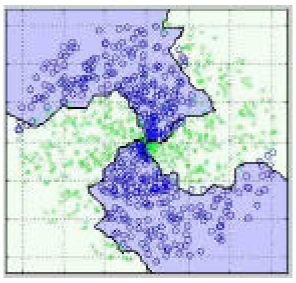
Paul Couchman
Fabien Imbault
Ronan Tigreat
Gorka Urchegui Tellechea

Classification assignment (group 6)
Image processing
MSc Embedded Systems
March 2003



Classification includes a broad range of decision-theoric approaches to the identification of images. All classification algorithms are based on the assumption that the image in question depicts one or more features and that each of these features belongs to one of several distinct and exclusive classes. The classes may be specified a priori through the use of training data or automatically clustered into sets of prototype classes.

In this assignment, we investigated the use of neural networks for supervised classification

We focused on the perceptron algorithm, then on the multilayer perceptron, and provide some matlab demonstrations.

INTRODUCTION TO CLASSIFICATION

with

NEURAL NETWORKS

Introduction

Image classification uses the ability to detect patterns in any given data set. To detect patterns (or classes) we need to define decision rules to categorise the data samples.
Classification can be supervised – where the system is output is evaluated by a teacher and classed, or unsupervised– where the algorithm /system itself attempts to create classes from the data which the user then attempts categorise into **meaningful** classes after the classification process.

The main techniques for pattern classification are summarised below, although in practice a combination of techniques are used for any given application.

1 - Statistical pattern classification

This method relies on defining a set of decision rules based on standard statistical theory. A set of characteristic feature measurements are extracted from the input image data and then combined to define a feature vector, which identifies the object as belonging to a pattern class. In this method the decision rules for object assignment are obtained using prior knowledge of the expected distribution of the pattern class or from results seen from test data applied (i.e. we already know what we are looking for to define a class).

In this method it can be difficult to express structured information

Statistical classification can take the form of **Template Matching** - the template is the ideal representation of the pattern in the image, and is correlated with the image under investigation. It can be applied to Optical Character Recognition (**OCR**) where use of a peep-hole template can overcome problems where a normal template fails due to a large portion of character area is common to many characters. The drawbacks of this method is that no variation in scale or orientation is possible between template and image – although this can be cured with pre-processing techniques such as normalisation. This method can also require large amounts of computation if small templates which focus

on key features are not used. The method is most suited to a controlled environment where the number of objects to be classified is not large.
Characters can also be classified from an analysis of their horizontal and vertical line structure, or according to features such as line endings, corners, junctions, and crossovers. This provides a method **independent** of character size.
A method mentioned in earlier course notes is the Decision Theoretic approach where a weighting vector is defined which will give an identifiable response with one class and a different response with another – a variant of this is the nearest neighbour technique.

2 – Syntactic pattern classification

This method relies on the underlying structure of the patterns themselves i.e. decompose (parse) a pattern into sub-patterns (or primitives). The recognition of each pattern is usually made by parsing the pattern structure according to syntax rules. A block diagram of the process is given overleaf-

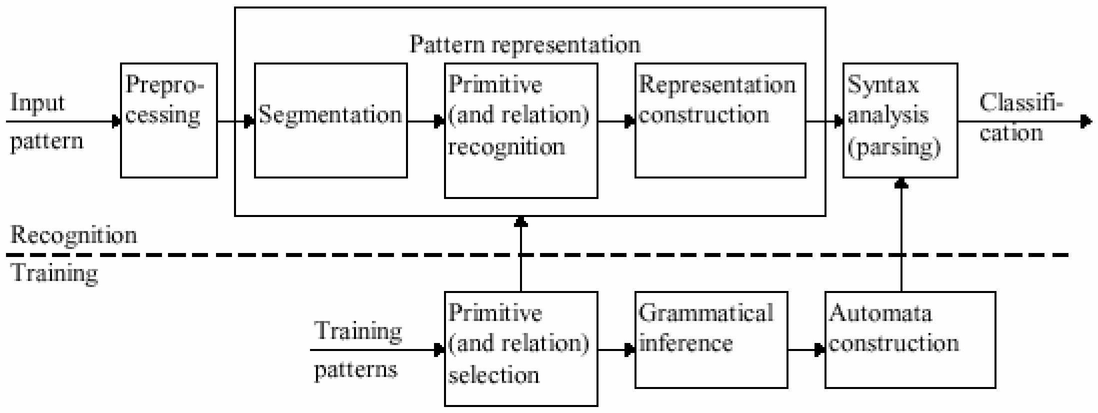


Fig 1 – Block Diagram of a Syntactic Pattern recognition System

3 – Neural Networks

The fundamental building block of a neural network is the neuron, which consists of a set of input connecting links, an adder, and an activation function.

Neurons can utilise a number of different types of activation function, such as a step function (known as a Hard Limiter) which allows only two discrete output values, and a sigmoid function which can allow a range of output values between an upper and a lower limit. Many other functions are possible and a range are shown below (Fig 2).

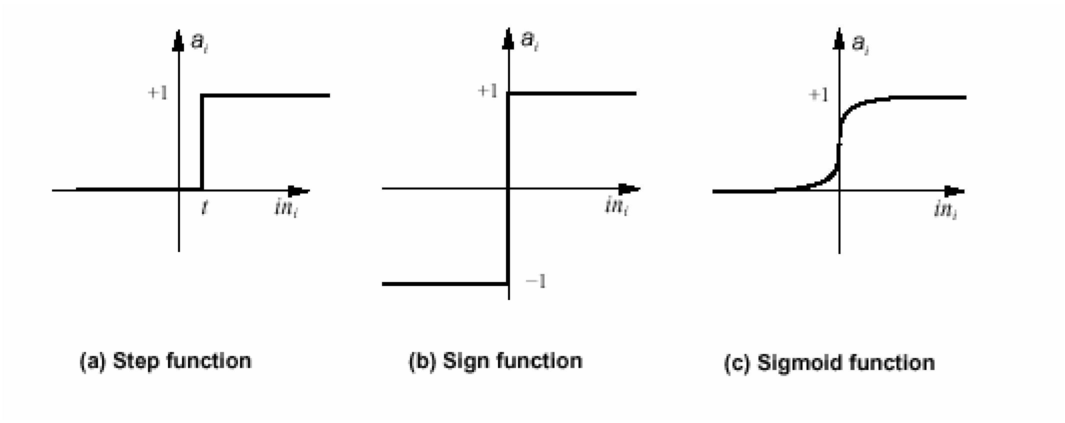


Fig 2 – Types of activation function

The basic neuron is known as the Perceptron which utilises the Step or Sign activation function, and gives rise to the term Threshold Logic Unit when combined with an input offset threshold. The block diagram is shown below:

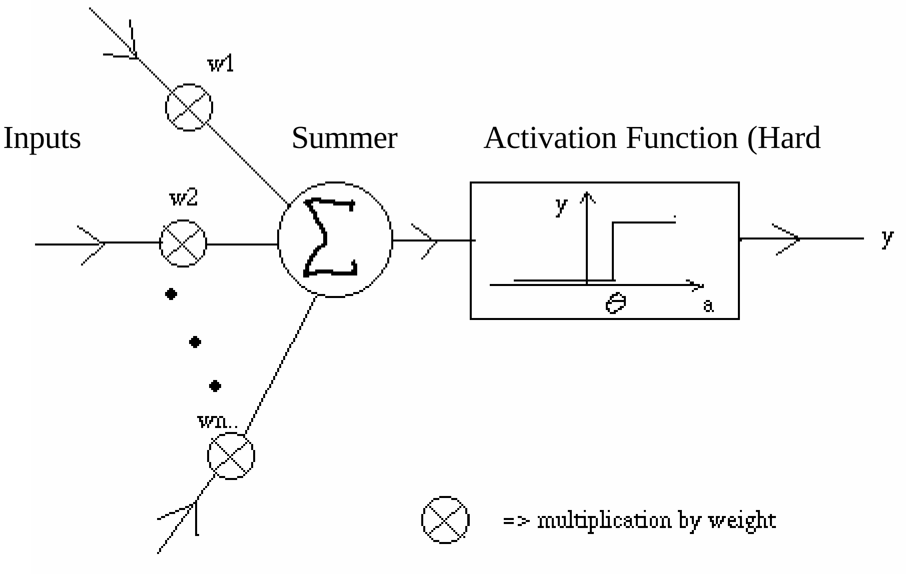


Fig 3 – Block Diagram of a Neuron (example Perceptron)

The input links are connected to weighting functions which are applied prior to presentation to the adder.

The adder then takes the weighted inputs and sums them. (The adder can also include an offset to raise or lower the net input to create a threshold).

The activation function is then applied to the result, thereby dictating what type of response the neuron makes to the summation.

An essential process in the creation of a Neural network is the training phase. During this process the weighting functions applied to the inputs are adjusted using an adaptive learning rule such as the Perceptron Convergence algorithm. In this case we say that if Class 1 and Class 2 are linearly separable (lie on opposite sides of a straight line in 2D or hyperplane in xD) it is possible to achieve a weight vector (Wt) such that

Wt x >=0 for each case of x belonging to Class 1
Wt x < 0 for each case of x belonging to Class 2

We then apply a training set which consists of the input vector x, together with the class membership (i.e. 1 or 0) of the input vector.
The training algorithm consists of two steps:

1. If the Kth member of the training set **x**k(k=1,2,..,K) is correctly defined by the weight vector Wk, on the Kth repeat of the calculation, we make no change to the value of Wk i.e.
   W(k+1)= Wk if Wk xk >=0 , and xk belongs to Class 1
   or W(k+1)= Wk if Wk xk < 0 , and xk belongs to Class 2

2. We update the weights if
   W(k+1)=Wk-(Q*xk) if Wk x Wt x >=0 for each case of x belonging to Class 2
   W(k+1)=Wk+Q*xk if Wk x Wt x < 0 for each case of x belonging to Class 1

   Where Q =learning step

Each repeat of the set of training data to the perceptron network is called an Epoch. The process is repeated until the weights stabilise, or max no. of iterations is reached.

Figure 4 demonstrates the practical limitations of the preceptron and the effect when it is presented with classes which are not linearly separable.

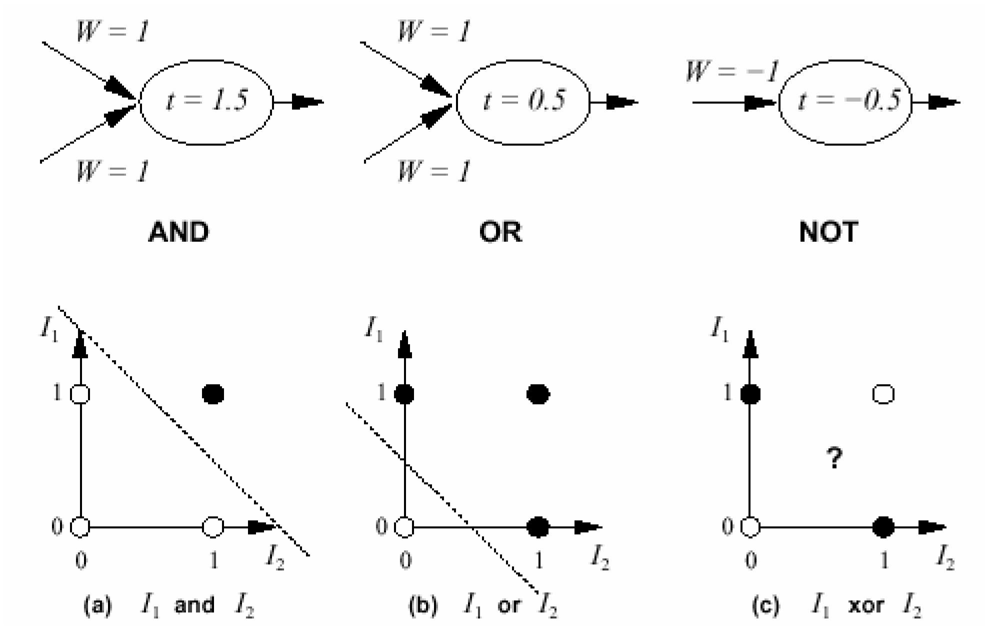
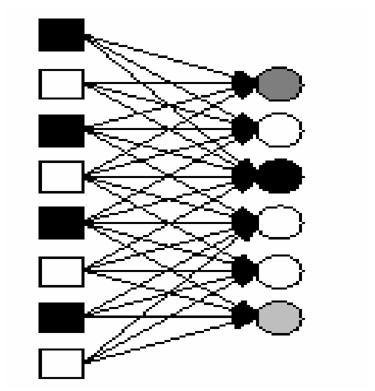


Fig 4 – The preceptrons ability to deal with logic evaluation

As can be clearly seen in case (c) it is not possible to separate the two classes linearly and the perceptron algorithm when confronted with this would iterate continuously or until a preset upper limit is reached

The topology of the network can dictate the ability to classify, and the range of classes it can cope with. Although the perceptron individually can only classify 2 classes (1 or 0) it can classify more classes when the network inputs are connected in parallel to each individual perceptron (in this case each perceptron represents a class)- See Fig 5. However this technique is limited when more than one input vector maps to multiple classes.



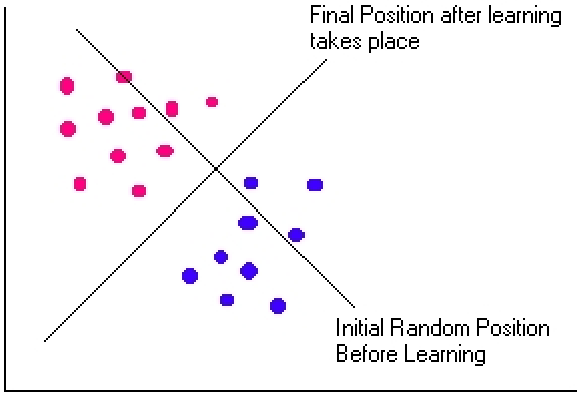
Inputs Outputs – each represents a different class

Fig 5– The topology of a Parallel Perceptron Network

MULTILAYER PERCEPTRON (MLP)

1 - Why using a MLP ?

The single layer perceptron can solve problems of linear separation like the one shown below:



Final Position after learning takes place
Initial Random Position Before Learning

Fig 6 – Linear separation with single layer perceptron

The problem with the single layer perceptron method is that it cannot solve linearly inseparable problems of the kind shown below:

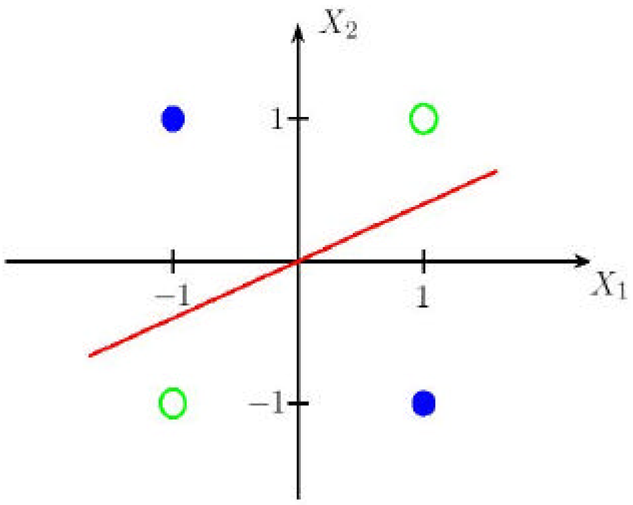


Fig 7 – XOR problem

The reason for this difficulty is due to the use of the step function as the thresholding process. The use of this function removes information that is vital to the network if it is going to learn successfully.

The information that is removed is the scale by which one needs to adjust the weights. This is due to neurons who are "barely ON" being treated the same as neurons who are "very much ON", when in reality they should be different. This problem is known as the *credit assignment problem*.

The way around this problem is to us a slightly different function that is non-linear (such as the sigmoid). Now our input is not simply on or off, but instead lies within a range. As a result of "smoothing" out the step function to a non linear function it shall turn on or off as before, but in the sloping region in the middle we will get further information about the weights and whether to increase/decrease them. This is the principle of the Multi-layer perceptron.

2 - Description of the MLP

The multi-layer perceptron neural network model consists of a network of processing elements or nodes arranged in layers. Typically it requires three or more layers of processing nodes: an input layer which accepts the input variables (e.g. satellite channel values, GIS data etc.) used in the classification procedure, one or more hidden layers, and an output layer with one node per class (figure below). The principle of the network is that when data from an input pattern is presented at the input layer the network nodes perform calculations in the successive layers until an output value is computed at each of the output nodes. This output signal should indicate which is the appropriate class for the input data i.e. we expect to have a high output value on the correct class node and a low output value on all the rest.

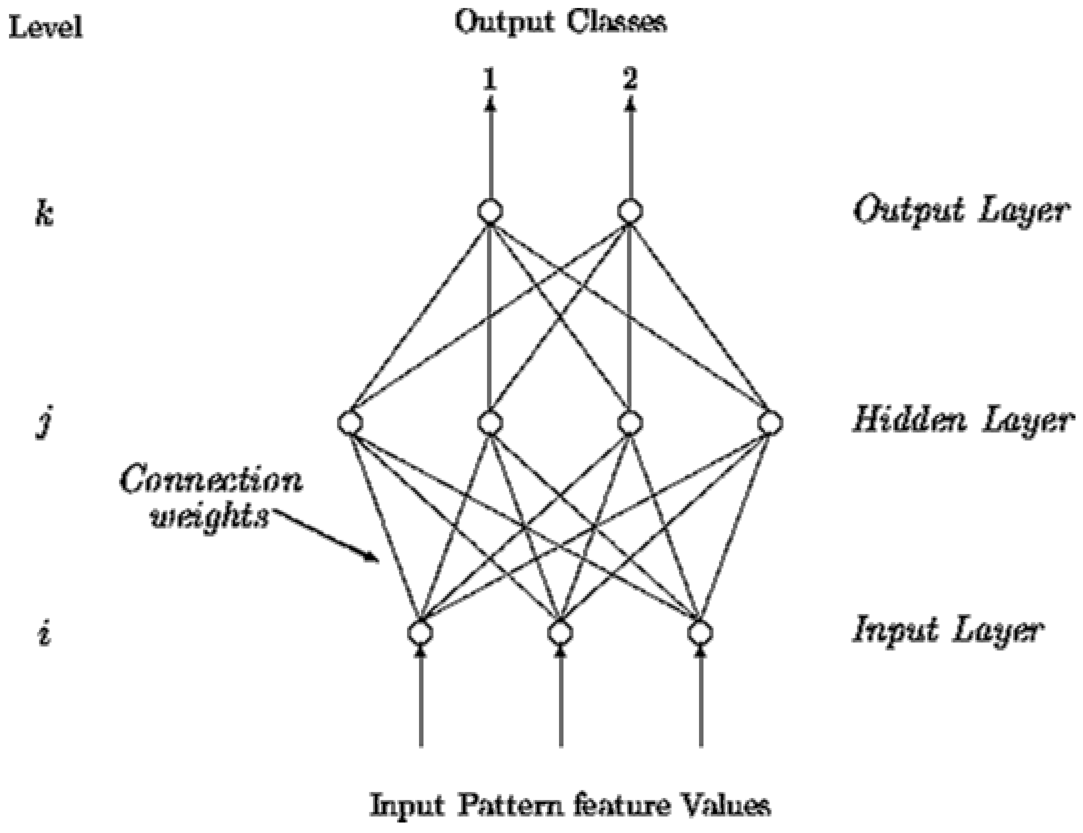


Fig 8 – Architecture of multi-layer perceptron

Every processing node in one particular layer is usually connected to every node in the layer above and below. The connections carry weights which encapsulate the behaviour of the network and are adjusted during training. The operation of the network consists of two stages. The ``forward pass" and the ``backward pass" or ``back-propagation". In the ``forward pass" an input pattern vector is presented to the network and the output of the input layer nodes is precisely the components of the input pattern. For successive layers the input to each node is then the sum of the scalar products of the incoming vector components with their respective weights.

That is the input to a node *j* is given by

$$input_j = \sum_i w_{ji} out_i$$

where w_{ji} is the weight connecting node *i* to node *j* and out_i is the output from node *i*.

The output of a node *j* is

$$out_j = f(input_j)$$

which is then sent to all nodes in the following layer. This continues through all the layers of the network until the output layer is reached and the output vector is computed. The input layer nodes do not perform any of the above calculations. They simply take the corresponding value from the input pattern vector.

The function *f* denotes the activation function of each node. A sigmoid activation function is frequently used,

$$f(x) = \frac{1}{1 + \exp(-x)}$$

where $x = input_j$. This ensures that the node acts like a thresholding device. The sigmoid curve is illustrated in next figure:

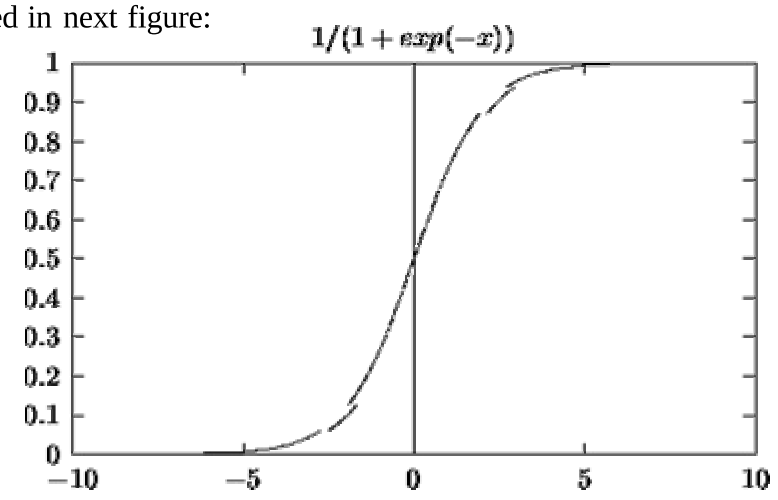


Fig 9 – The sigmoid activation function.

The multi-layer feed-forward neural network is trained by supervised learning using the iterative back-propagation algorithm. In the learning phase a set of input patterns, called the training set, are presented at the input layer as feature vectors, together with their corresponding desired output pattern which usually represents the classification for the input pattern. Beginning with small random weights, for each input pattern the network is required to adjust the weights attached to the connections so that the difference between the network's output and the desired output for that input pattern is decreased. Based on this difference the error terms for each node in the output layer are computed. The weights between the output layer and the layer below (hidden layer) are then adjusted by a rule called the *generalised delta rule* (Rumelhart)

$$w_{kj}(t+1) = w_{kj}(t) + \eta(\delta_k out_k)$$

where $w_{kj}(t + 1)$ and $w_{kj}(t)$ are the weights connecting nodes k and j at iteration (t+1) and t respectively,   is a learning rate parameter. Then the   terms for the hidden layer nodes are calculated and the weights connecting the hidden layer with the layer below (another hidden layer or the input layer) are updated. This procedure is repeated until the last layer of weights has been adjusted.

The □term in equation above is the rate of change of error with respect to the input to node k, and is given by

$$\delta_k = (d_k - out_k) f'(input_k)$$

for nodes in the output layer, and

$$\delta_j = f'(input_k) \sum_k \delta_k w_{kj}$$

for nodes in the hidden layers, where d_k is the desired output for a node k.

The back-propagation algorithm is a gradient descent optimisation procedure which minimises the mean square error between the network's output and the desired output for all input patterns P

$$E = \frac{1}{2P} \sum_p \sum_k (d_k - out_k)^2$$

The training set is presented iteratively to the network until a stable set of weights is achieved and the error function is reduced to an acceptable level. The next figure summarises the training procedure of the multi-layer feed-forward neural network. To measure the generalisation ability of the multi-layer feed-forward neural network it is common to have a set of data to train the network and a separate set to assess the performance of the network during or after the training is complete. Once the neural network has been trained, the weights are saved to be used in the classification phase. During classification, image data are fed into the network which performs classification by assigning a class number to a pixel or segment using the numerical values computed at the output layer. Typically the output node giving the highest value is taken as the best class to assign.

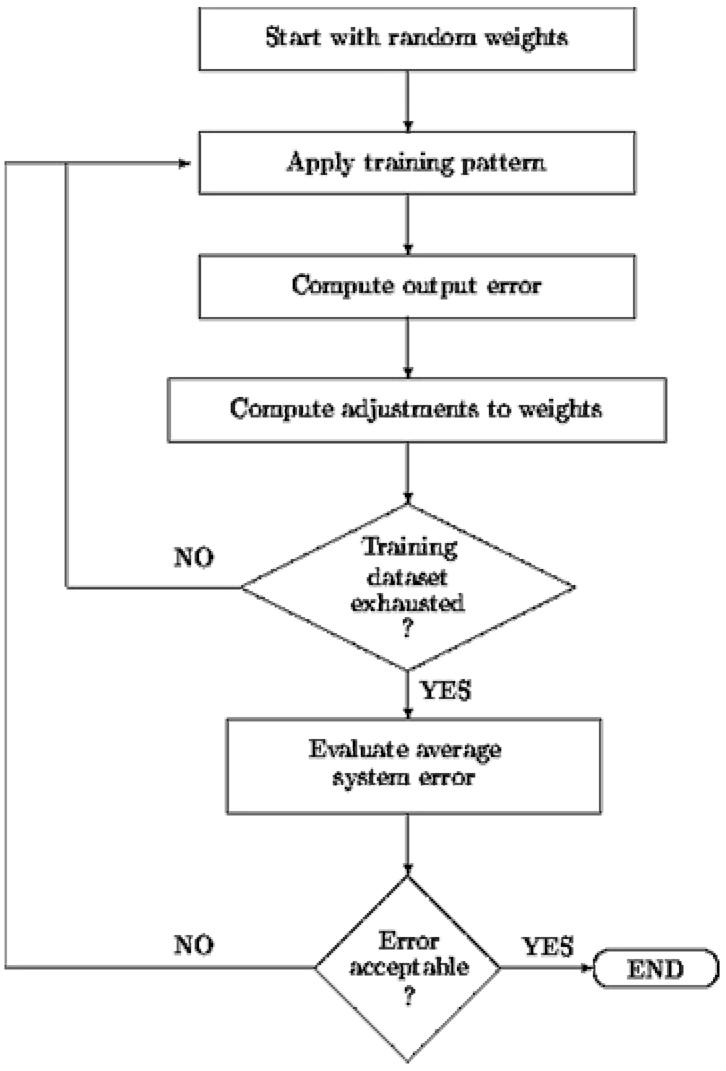

Start with random weights
Apply training pattern
Compute output error
Compute adjustments to weights
NO
Training dataset exhausted ?
YES
Evaluate average system error
NO
Error acceptable ?
YES
END

Fig 10 – Training procedure for multi-layer perceptron network

When showing the untrained network an input pattern, it produces a random output. We need to define an error function which represents the different between our output received and the output we desire. This error function must be continually reduced so our output reaches that of the desired output.

To achieve this we adjust the weights on the links between layers, this is done by the generalised delta rule calculating the error for a particular input and back-propagating it through to the previous layer. Thus each unit in the network has its weights adjusted so that it reduces the value of the error function and the network learns. The MLP is thus able to distinguish more complex pattern :

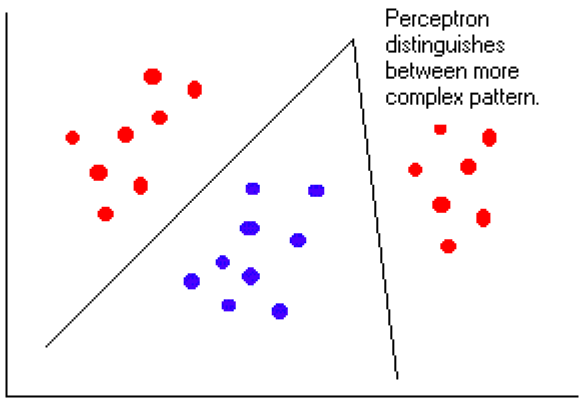


Fig 11 – Complex pattern classification

3 - Summary of MLPs

The MLP is the most widely used neural classifier today. It belongs to the class of supervised neural networks. MLP networks are general-purpose, flexible, non linear models consisting of a number of units organised into multiple layers. The complexity of the MLP network can be changed by varying the number of layers and the number of units in each layer. Given enough hidden units and enough data, it has been shown that MLPs can approximate virtually any function to any desired accuracy. In other words, MLPs are universal approximators. MLPs are valuable tools in problems when one has little or no knowledge about the form of the relationship between input vectors and their corresponding outputs.

HANDWRITTEN DIGIT RECOGNITION

The aim is to illustrate with a practical example the different steps in the development of a classifier. The chosen classification problem is the recognition of handwritten digits.

There are 10 different digits, so the classifier has 10 output classes. It is a complicate classification problem because each person writes the digits differently. It is important to choose good features to be able to classify correctly the different digits, and train the classifier with a large amount of digits.

Between the different techniques to classify, a neuronal net is chosen for the handwritten digits recognition. Neuronal nets are widely used in classification problems as was mentioned in the lectures. Matlab neuronal net toolbox is used to train and simulate the net.

These are the steps to develop the classifier:

1.- Search of the handwritten digits
2.- Preprocessing of the handwritten digits
3.- Feature extraction
4.- Training of the Neuronal Net
5.- Testing the Neuronal Net

<u>1.- Search of the handwritten digits</u>

110 images of digits have been used to develop the classifier, 11 example images for each class. Some have been downloaded from Internet and the others have been created scanning handwritten digits. All the figures have the same size, 40x30 pixels, and they can be in colour or in black and white.

The figure 12 shows some of the used digits:

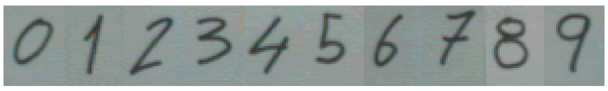


Fig 12.- Some Digits to recognise

<u>2.- Preprocessing of the handwritten digits</u>

To extract the features from the images, they have to be binary images, in black and white. That is the only preprocessing that the images need because there is not noise in the digits and is not necessary to apply filters.

The first step is to convert the colour image to grey level image. After, the histogram is calculated and the image is threshold. The Otsu technique is used to decide the optimal threshold value for each image, using the function given in the Matlab tutorials.

The figure 13 shows some digits before and after the preprocessing:

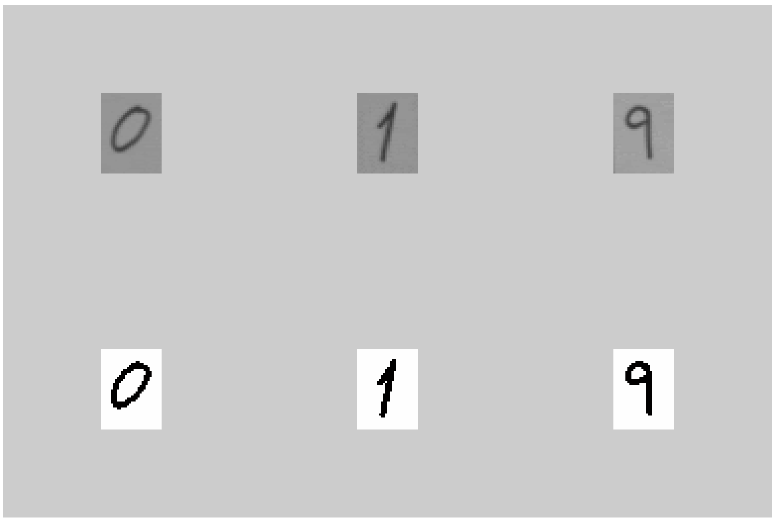


Fig 13.- Digits before and after the preprocessing

Not all the handwritten digits are located in the centre of the image, so it is necessary to detect where the digit exactly is to obtain better features. Once the image is threshold, the boundaries of the digit are located and the features are calculated using the pixels inside the boundaries.

The figure 14 shows the boundaries for one of the digits. Only the pixels inside the rectangle are used.

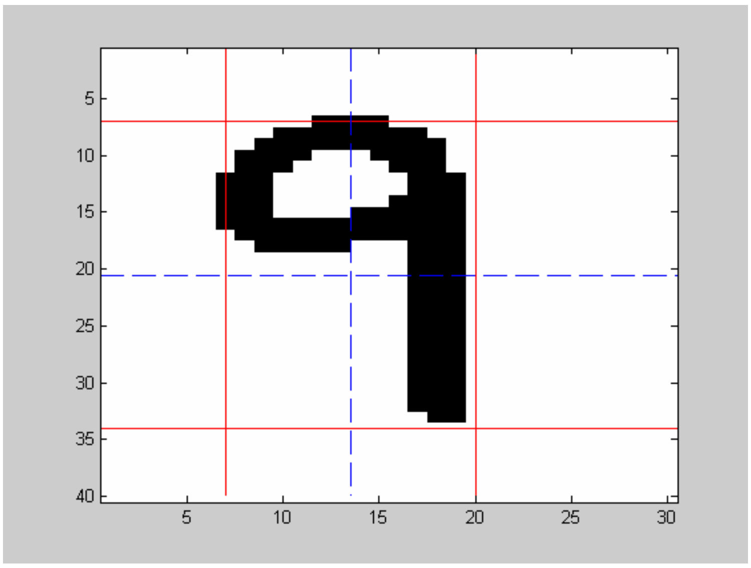


Fig 14.- Boundaries of the digit 9

Using the boundaries, the high and the width of the digits is calculated. These two values are used to normalize the distance and this way the features are independent on the size of the digit. Also the central point of the digit is calculated and some of the features are calculated respect this point.

3.- Feature extraction

8 different features are used to classify the handwritten digits:

Feature 1:

The first feature is the relation between the high and the width of the handwritten digit. The high and width are calculated in the preprocessing.

feature 1 = high_digit / width_digit;

Feature 2 and Feature 3:

These two features check how the black pixels are distributed in the rectangle. First the number of black pixels inside the rectangle is calculated. (total_pixels of the digit).

The feature 2 is the percentage of pixels located in the upper area of the digit, in other words, the pixels located up the central point:

feature 2 = upper_pixels/ total_pixels;

The feature 3 is the percentage of pixels located in the left area of the digit, in other words, the pixels located in the left of the central point:

feature 3 = left_pixels/ total_pixels;

Feature 4:

The next feature is the average of the distance between all the black pixels and the central point.

Feature 4 = $\frac{1}{total\_pixels} \times \sum_y \sum_x \sqrt{(x-i)^2 \times (y-j)^2}$

Where (x,y) are the coordinates of a point
and (i,j) are the coordinates of central point.

Feature 5, 6, 7, 8:

The last 4 features calculate the moments of the digit respect the central point.

For a f(x,y) two dimensional function, the central moment of order (p + q) is defined as:

$$m_{pq} = \frac{1}{total\_pixels} \sum_y \sum_x (x-i)^p \times (y-j)^q \cdot f(x,y)$$

Where (x,y) are the coordinates of a point
and (i,j) are the coordinates of central point.

In the handwritten digit recognition, the 4 moments of order 3 are used as features. These are the moments of order 3:

M12:

$$m_{12} = \frac{1}{total\_pixels} \sum_y \sum_x (x-i) \times (y-j)^2 \cdot f(x,y)$$

M21:

$$m_{21} = \frac{1}{total\_pixels} \sum_y \sum_x (x-i)^2 \times (y-j) \cdot f(x,y)$$

M03:

$$m_{03} = \frac{1}{total\_pixels} \sum_y \sum_x (y-j)^3 \cdot f(x,y)$$

M30:

$$m_{30} = \frac{1}{total\_pixels} \sum_y \sum_x (x-i)^3 \cdot f(x,y)$$

4.- Training of the Neuronal Net

There are 110 different images of the digits, 11 images per class. A hundred of these images are used to train the neuronal net and the other 10 to test the performance of the network. Most of the images are used to train the network because is important to train with a large amount of digits.

The output layer needs 10 neurons, one per each class to recognize. Different neuronal nets can be chosen to recognize the handwritten digits. Neuronal nets with different number of layers and different neurons per layer have been trained and simulated. Also different transfer functions have been implemented and their performance has been compared.

The neuronal net with better performance is shown in the figure 15:

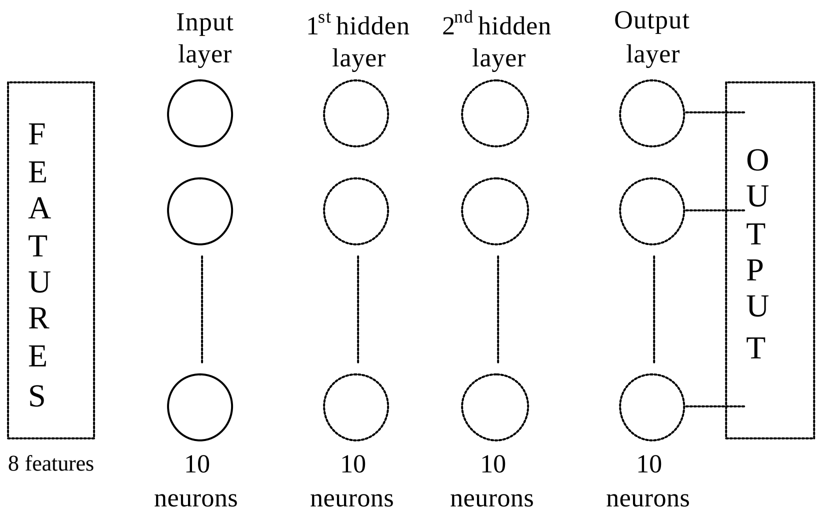


Fig 15: The neuronal network to classify the digits

There are 4 different layers and each one has got 10 neurons. The neurons of the input layer have got 8 inputs that are the calculated features. All the others neurons have got 10 inputs that are the outputs of the previous layer.

All the neurons use the hyperbolic tangent sigmoid or tansig transfer function. This transfer functions has been chosen because the performance of the network is better and less neurons and layers are needed to recognise the handwritten digits.

If the number to recognize is a zero, the value of the first output neuron a 1, and the output of all the others neurons is a zero. The same happens for the others digits.

Matlab has got different algorithms to train the network. In this case the gradient descent weight and bias learning function is used. The 'learngd' function adjusts the weight and bias of the different layers depending on the gradient in the output error. The objective is to minimize the Sum squared error in the output.

The figure 16 shows the training of the network. At the beginning the Sum squared error (sse) is big, but after 300 epochs the error of the network is very small and the neuronal net can recognize the training digits with a very small error.

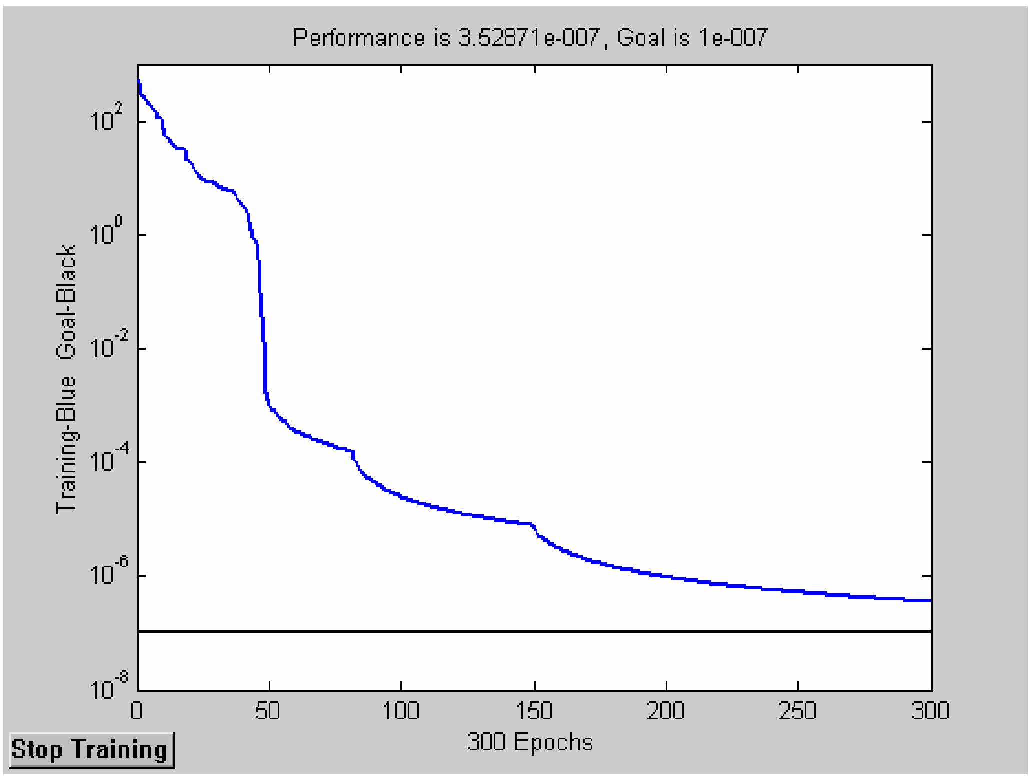


Fig 16.- Training of the neuronal network

5.- Testing the Neuronal Net

Once the neuronal net has been trained, the performance can be calculated using Matlab functions.

First the neuronal network is simulated with the training images. The 100 digits used to train the network are correctly classified, all the digit are classified in the correct class.

Now the network is simulated with the 10 images that are not used in the training. There is an image for each class. The table 1 shows the result of the simulation, in grey colour are shown the number that have been bad classified:

Digit to recognise	0	1	2	3	4	5	6	7	8	9
Digit classified has	8	1	5	3	4	5	9	7	8	9

Table 1.- Result of the simulation

Three digits have been bad classified, the 0 is recognized as a 8, the 2 is recognized as a 5 and the 6 is recognized as a 9. All the other digits are recognized correctly.

Only 10 elements for each class have been used to train network, and using more digits to train the neuronal network the result should be better. Another way to improve the result is using other features.

The feature selection and extraction is the important and hard part of the classification because the training and simulation of the network is quite easy to implement using the Matlab function. The performance of the classifier depends more on the chosen features than on the used neuronal network configuration.

HOW TO USE THE MATLAB FILES:

Matlab neuronal network is necessary to train and simulate the network

* First execute the ‘training_files.m’ file:
    - This file uses the functions that calculate the features and saves all the features in the files ‘training_features.txt’, ‘training_targets.txt’, ’simulation_features.txt’, and ‘simulation_targets.txt.’

* Execute the ‘network_training.m’ to train the network.

* Execute the ‘simulate_network.m’ file to simulate the network. The results are saved in the files ‘simulation_outputs.txt’ and ‘simulation_training_data.txt’.

MATLAB TUTORIAL

1 - Perceptron demo : classifying 3D patterns into 2 classes

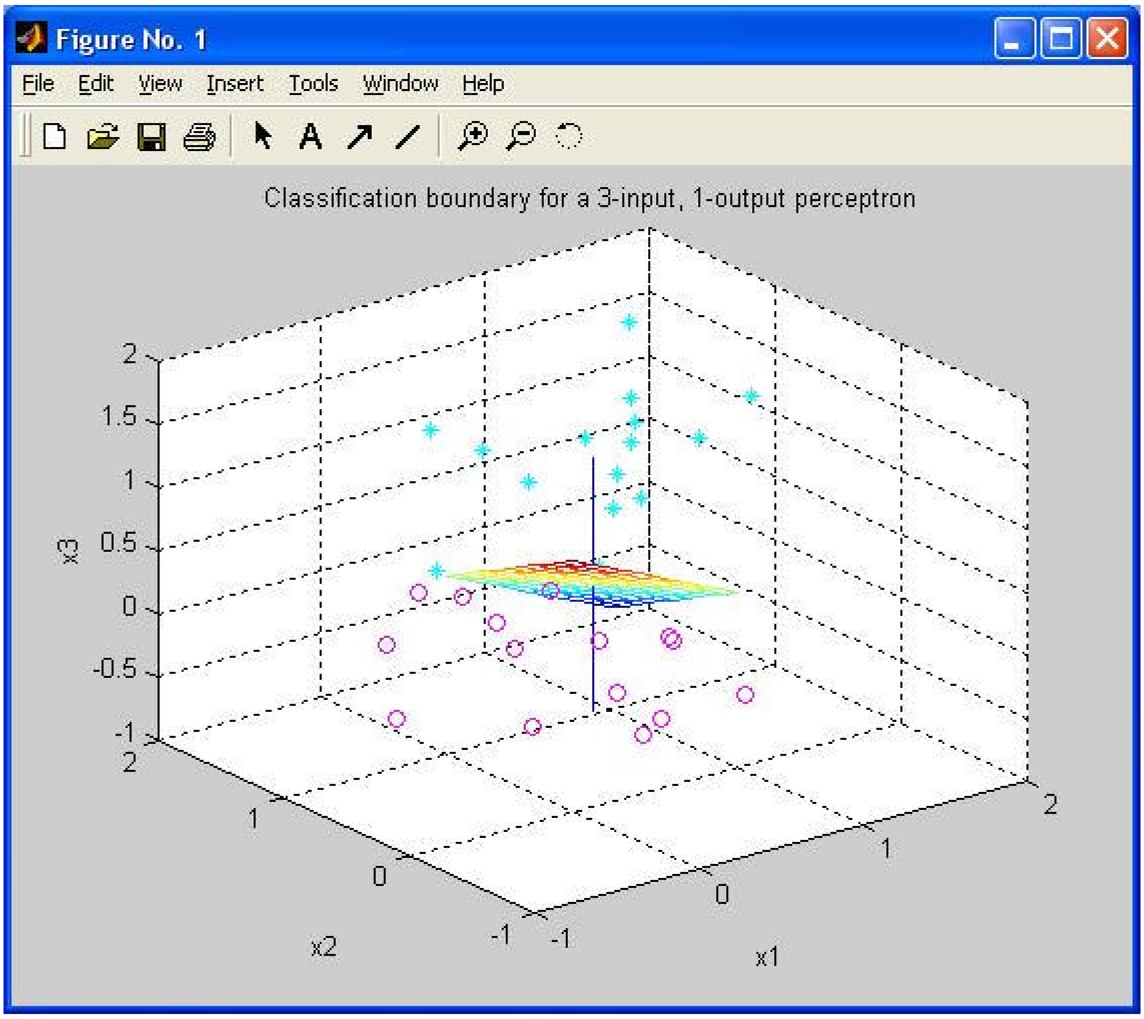


Fig 17 – Perceptron Demo

Download:
initPerceptron.m (initialization module)
plotInput.m (plots the input patterns)
plotDecisionSurf.m (plot the weight vector, and the classification boundary defined by the weight vector)
trainPerceptron.m (function for implementing the learning)
runPerceptron.m (calls the plotting m-files and trainPerceptron)
demoPerceptron.m (calls initPerceptron and runPerceptron until convergence or maximal number of iterations)

Principle:

The perceptron which has been implemented is a simple feedforward algorithm with linear treshold activation function. If the total weighted summed to a unit is greater than or equal to the threshold, its state is set to 1, otherwise it is -1.

The following rule has been used: after a pattern is presented to the network, we compare the value of the actual output produced by the network to the value of the target output. If the target is +1 and output is -1, we update the weight vector according to :

$$weights = weights + learning\_rate \times input\_vector$$

Otherwise, if unit should be -1 and is actually +1, update the weight vector according to

$$weights = weights - learning\_rate \times input\_vector$$

If the patterns are linearly separable, then the perceptron learning algorithm will converge on a perfect solution.
Note that if the patterns are not linearly separable, the algorithm does not converges.

Questions:

a) Have a look at the code in 'initPerceptron.m', 'runPerceptron.m', 'trainPerceptron.m' and 'demoPerceptron.m' to see how the weights and states are initialized, and how the weights are updated. Now do the following:
   Execute the script 'demoPerceptron, which initializes the weights and training patterns by calling 'initPerceptron' and then repeatedly run the script 'runPerceptron' until the network has converged (weights no longer change) or until the maximal number of iterations has been reached. You can view the weight vector, and the associated classification boundary (desicion surface perpendicular to the weight vector) on the figure.
   Repeat this entire process for several runs, using different random patterns and different random initial weights. Why does the convergence time differ from one run to the next?

b) Increase the difficulty of the classification problem by changing the variance of the random Gaussian function used to generate the training patterns. Does the Perceptron converge more quickly or more slowly on these data, or does it converge at all? How can you explain the convergence or lack of convergence by looking at the training patterns? Conclude on the limitations of the perceptron.

2 - Matlab Classification Toolbox (Copyright John Wiley & Sons)

Download:

The classification toolbox can be freely downloaded from the internet (http://tiger.technion.ac.il/~eladyt/Classification_toolbox.html). To install it, you just need to unzip the file and specify the path (use File|set Path). A user guide is provided with the software.

Principle:

You have access to a large number of classification algorithms that you can vizualise the effects of different parameters: the input data, the error estimation method, the percentage of training vectors, the use of a preprocessing step... A multiple algorithm comparison tool is provided.

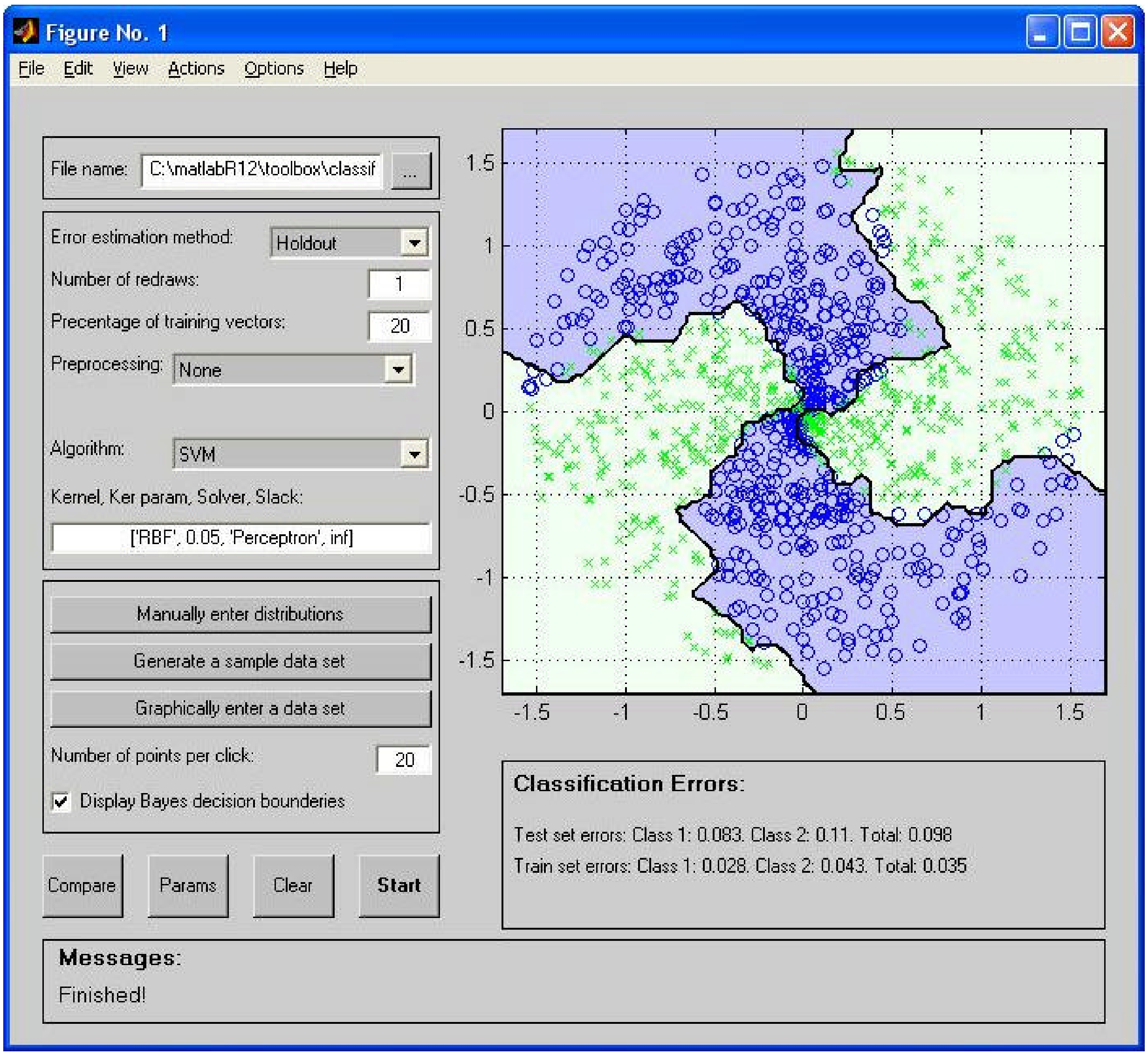


Figure No. 1
File Edit View Actions Options Help
Manually enter distributions
Generate a sample data set
Graphically enter a data set
Compare
Params
Clear
Start
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5

File name: C:\matlabR12\toolbox\classif

Error estimation method: Holdout

Number of redraws: 1

Precentage of training vectors: 20

Preprocessing: None

Algorithm: SVM

Kernel, Ker param, Solver, Slack: ['RBF', 0.05, 'Perceptron', inf]

Number of points per click: 20

☑ Display Bayes decision bounderies

Classification Errors:

Test set errors: Class 1: 0.083. Class 2: 0.11. Total: 0.098

Train set errors: Class 1: 0.028. Class 2: 0.043. Total: 0.035

Messages:

Finished!

Fig 18 – Classification Toolbox

Questions:

a) Open the file called XOR. Choose different algorithms and try to train the data. Verify that a perceptron cannot classify non-linearly separable data. Try to use a backpropagation algorithm. What is the influence of preprocessing on the results ?
b) Using the Holdout error estimation option, evaluate the influence of the percentage of training vectors. What difference do you make between training vectors and test vectors ?
c) Get some information on cross-validation. Using cross-validation allow us to reduce the percentage of training vectors. Explain why.