Master of Science Thesis

Spam Classification Using Machine Learning Techniques - Sinespam

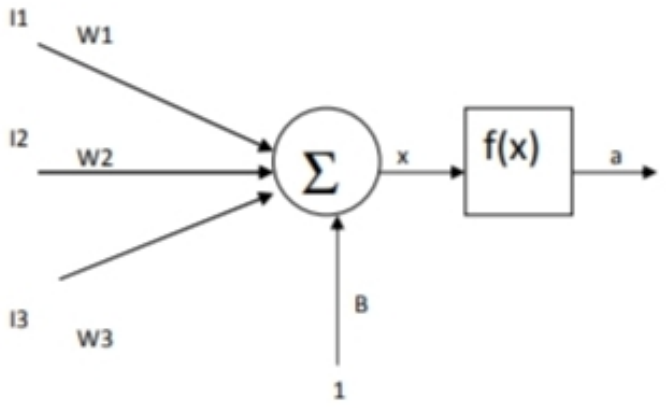
José Norte Sosa

5 - Artificial Neural Network Overview

An artificial neural network is a collection of connected neurons. Taken one at a time each neuron is a rather simple. As a collection however, a group of neurons is capable of producing complex results. In the following sections we will briefly summarize mathematical models of a neuron, neuron layer and neural network before discussing the types of behavior achievable from a neural network.

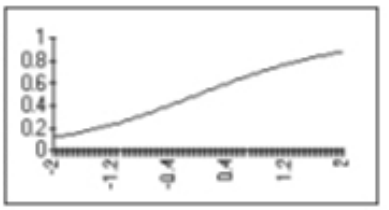
5.1 - Artificial Neuron Modeling

A model of a neuron has three basic parts: input weights, a summer or aggregation function, and an output function. The input weights scale values used as inputs to the neuron, the summer adds all the scaled values together, and the output function produces the final output of the neuron. Often, one additional input, known as the bias is added to the model. If a bias is used, it can be represented by a weight with a constant input of one. This description is laid out visually below.



where **I1**, **I2**, and **I3** are the inputs, **W1**, **W2**, and **W3** are the weights, **B** is the bias, **x** is an intermediate output, and **a** is final output. The equation for **a** is given by $a = f(W_1 I_1 + W_2 I_2 + W_3 I_3 + B)$ where f could be any function. Most often, f is the sign of the argument (i.e. 1 if the argument is positive and -1 if the argument is negative), linear (i.e. the output is simply the input times some constant factor), or some complex curve used in function matching e.g. Logistic or hyperbolic tangent.

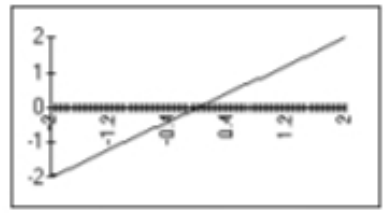
Logistic (Sigmoid logistic) - We have found this function useful for most neural network applications. It maps values into the (0, 1) range. Always use this function when the outputs are categories.



Logistic Function

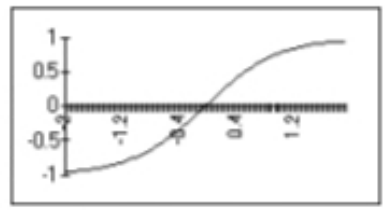
Linear - Use of this function should generally be limited to the output slab. It is useful for problems where the output is a continuous variable[Jordan, 1995], as opposed to several outputs which represent categories. Stick to the logistic for categories. Although the linear

function detracts from the power of the network somewhat, it sometimes prevents the network from producing outputs with more error near the min or max of the output scale. In other words the results may be more consistent throughout the scale. If you use it, stick to smaller learning rates, momentums, and initial weight sizes. Otherwise, the network may produce larger and larger errors and weights and hence not ever lower the error. The linear activation function is often ineffective for the same reason if there are a large number of connections coming to the output layer because the total weight sum generated will be high.



Linear Function

Tanh (hyperbolic tangent) - Many experts feel this function should be used almost exclusively. It is sometimes better for continuous valued outputs, however, especially if the linear function is used on the output layer. If you use it in the first hidden layer, scale your inputs into [-1, 1] instead of [0,1]. We have experienced good results [McCullagh, P. and Nelder, J.A. (1989)] when using the hyperbolic tangent in the hidden layer of a 3 layer network, and using the logistic or the linear function on the output layer.



Tanh Function

When artificial neurons are implemented, vectors are commonly used to represent the inputs and the weights so the first of two brief reviews of linear algebra is appropriate here. The dot product of two vectors $\vec{x} = (x_1, x_2, \cdots, x_n)$ and $\vec{y} = (y_1, y_2, \cdots, y_n)$ is given by $\vec{x} \bullet \vec{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$. Using this notation the output is simplified to $a = f(\vec{W} \bullet \vec{I} + B)$ where all the inputs are contained in $\vec{I}$ and all the weights are contained in $\vec{W}$.

5.2 - Neuron Layer

In a neuron layer each input is tied to every neuron and each neuron produces its own output. This can be represented mathematically by the following series of equations:

$$a_1 = f_1(\vec{W}_1 \bullet \vec{I} + B_1)$$

$$a_2 = f_2(\vec{W}_2 \bullet \vec{I} + B_2)$$

$$a_3 = f_3(\vec{W}_3 \bullet \vec{I} + B_3)$$

$$\ldots$$

NOTE: In general these functions may be different.

And we will take our second digression into linear algebra. We need to recall that to perform the operation of matrix multiplication you take each column of the second matrix and perform the dot product operation with each row of the first matrix to produce each element in the result. For example the dot product of the *i*th column of the second matrix and the *j*th row of the first matrix results in the (*j*,*i*) element of the result. If the second matrix is only one column, then the result is also one column.

Keeping matrix multiplication in mind, we append the weights so that each row of a matrix represents the weights of an neuron. Now, representing the input vector and the biases as one column matrices, we can simplify the above notation to:

$$\vec{a} = f(W \cdot I + B)$$

which is the final form of the mathematical representation of one layer of artificial neurons.

5.3 – Multilayer Feed-Forward Neural Network

A multilayer feed-forward neural network is simply a collection of neuron layers where the output of each previous layer becomes the input to the next layer. So, for example, the inputs to layer two are the outputs of layer one. In this exercise we are keeping it relatively simple by not having feedback (i.e. output from layer *n* being input for some previous layer). To

mathematically represent the neural network we only have to chain together the equations. The final equation for a three layer network is given by:

$$a = f(W_3 \cdot f(W_2 \cdot f(W_1 \cdot \vec{I} + \vec{B}_1) + \vec{B}_2) + \vec{B}_3)$$

5.4 - Neural Network Behavior

Although transistor now switch in as little as 0.000000000001 seconds and biological neurons take about .001 seconds to respond we have not been able to approach the complexity or the overall speed of the brain because of, in part, the large number (approximately 100,000,000,000) neurons that are highly connected (approximately 10,000 connections per neuron). Although not as advanced as biologic brains, artificial neural networks perform many important functions in a wide range of applications including sensing, control, pattern recognition, and categorization. Generally, networks (including our brains) are trained to achieve a desired result. The training mechanisms and rules are beyond the scope of this work, however it is worth mentioning that generally good behavior is rewarded while bad behavior is punished. That is to say that when a network performs well it is modified only slightly (if at all) and when it performs poorly larger modifications are made. As a final thought on neural network behavior, it is worth noting that if the output functions of the neurons are all linear functions, the network is reducible to a one layer network. In other words, to have a useful network of more than one layer we must use a function like the sigmoid (an s shaped curve), a linear function, or any other non-linear shaped curve.

5.5 - Classification Problem – Training Networks

This section introduces the concept of training a network to perform a given task. Some of the ideas discussed here will have general applicability, but most of the time refer to the specific of TLUs (Threshold Logical Unit) to perform a given classification the network must implement the desired decision surface. Since this is determined by the weight vector and threshold, it is necessary to adjust these to bring about the required functionality. In general terms, adjusting the weights and thresholds in a network is usually done via an iterative process of repeated presentation of examples of the required task. At each presentation, small changes are made to weights and thresholds to bring them more in line with their desired value. This process is known as ***training*** the net, and the set of examples as the training set. From the network's viewpoint it undergoes a process of learning, or adapting to, the training set, and the prescription for how to change the weights at each step is the learning rule. In one type of training the net is presented with a set of input patterns or vectors $\{x_i\}$ and, for each one, a corresponding desired output vector or target $\{t_i\}$. Thus, the net is supposed to respond with t_k, given input x_k for every k. This process is referred to as supervised training (or learning) because the network is told or supervised at each step as to what it is expected to do.

Next there is a description, of how to train the TLUs and a related node, the perceptron, using supervised learning. This will consider a single node in isolation at first so that training set

consists of a set of pairs {v, t}, where v is an input vector and t is the target class or output ("1" or "0") that belongs to.

The first step toward understanding neural nets is to abstract from the biological neuron, and to focus on its character as a threshold logic unit (TLU). A TLU is an object that inputs an array of weighted quantities, sums them, and if this sum meets or surpasses some threshold, outputs a quantity. Let's label these features. First, there are the inputs and their weights: X1,X2, ..., Xn and W1, W2, ...,Wn. Then, there are the Xi*Wi that are summed, which yields the activation level a, in other words:

Â a = (X1 * W1)+(X2 * W2)+...+(Xi * Wi)+...+ (Xn * Wn)

The threshold is called theta. Lastly, there is the output: y. When a >= theta, y=1, else y=0. Notice that the output doesn't need to be discontinuous, since it could also be determined by a squashing function, s (or sigma), whose argument is a, and whose value is between 0 and 1. Then, y=s(a).

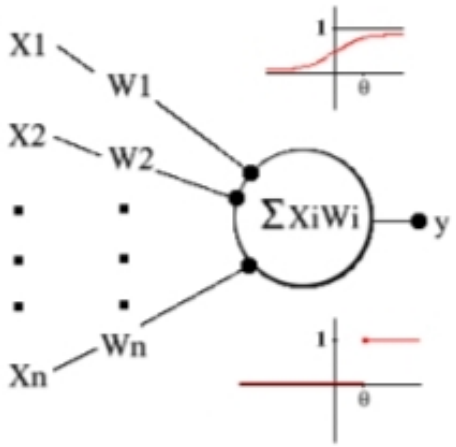


Figure 5.5.A Threshold logic unit, with sigma function (top) and cutoff function (bottom)

5.6 - Training the threshold as a weight

In order to place a adaptation of the threshold on the same footing as the weights, there is a mathematical trick we can play to make it look like weight. Thus, we normally write $w \cdot x \geq \Theta$ as the condition for output of a "1". Subtracting Θ from both sides gives $w \cdot x - \Theta \geq 0$ and making the minus sign explicit results in the form $w \cdot x + (-1)\ \Theta \geq 0$. Therefore, we may think of the threshold as an extra weight that is driven by an input constantly tied to the value -1. This leads to the negative of the threshold begin referred to sometimes as the bias. The weight vector, which was initially of dimension n for an n-input unit, now becomes the (n+1)-dimensional vector $w_1, w_2, \ldots\ldots w_n, \Theta$. We shall call this the augmented weight vector, in contexts where confusion might arise, although this terminology is by no means standard. Then for all TLUs we may express the node function as follows:

$$w \cdot x \geq 0 \quad \rightarrow \quad y = 1$$

$$w \cdot x < 0 \quad \rightarrow \quad y = 0$$

Putting $w \cdot x = 0$ now defines the decision hyperplane, which, is orthogonal to the(augmented) weight vector. The zero-threshold condition in the augmented space means that the hyperplane passes through the origin, since this is only way that allows $w \cdot x = 0$. We now illustrate how this modification of pattern space works with an example in 2D.

Example:

Consider two-input TLU that ouputs a "1" with input (1,1) and a "0" for all other binary inputs so that a suitable (non-augmented) weight vector is (1/2, 1/2) with threshold 3 / 4. This is shown next in figure 5.6.A where the decision line and weight vector are displayed. That the decision line goes through the points $x_1 = (1/2,1)$ and $x_2 = (1, 1/2)$ maybe easily verified since according to $w \cdot x_1 = w \cdot x_2 = 3/4 = \Theta$. For the augmented pattern space we have to go 3D as shown in next figure 5.6.B.

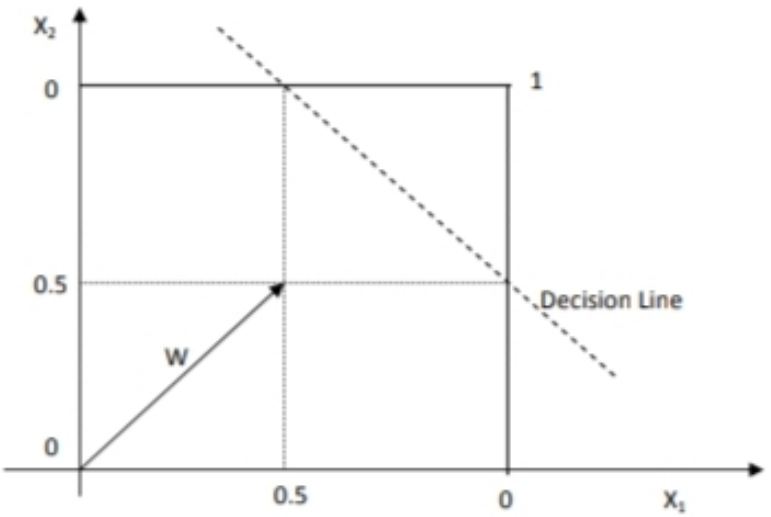


Figure 5.6.A – Two-dimensional TLU example

The previous two components x_1,x_2 are now drawn in the horizontal plane while third component x_3 has been introduced, which is shown as the vertical axis. All the patterns to the TLU now have the form (x_1, x_2, -1) since the third input is tied to the constant value of -1. The augmented weight vector now has a third component equal to the threshold and is perpendicular to a decision plane that passes through the origin. The old decision line 2D is formed by the intersection of the decision plane and the plane containing the patterns.

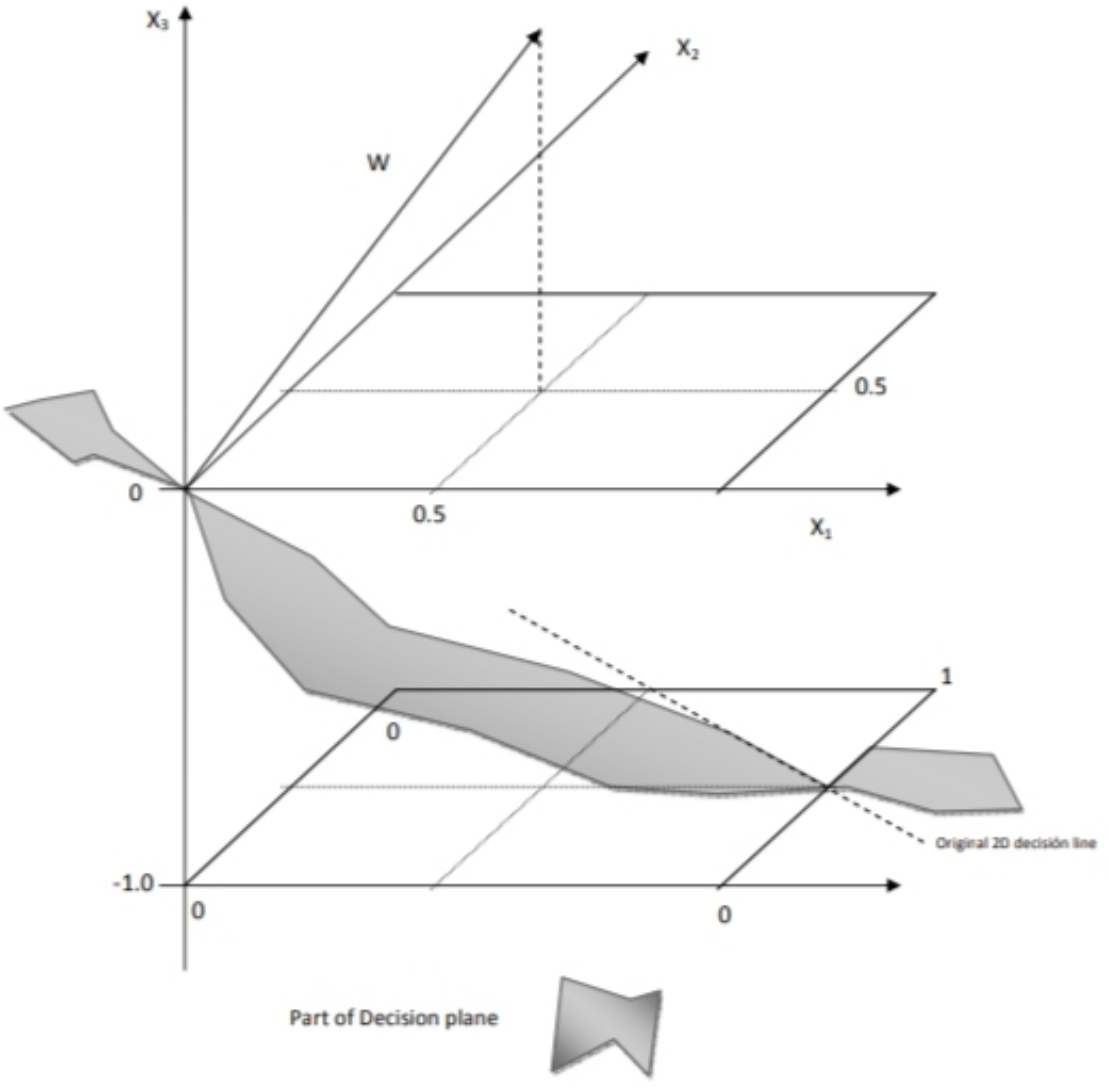


Figure 5.6.B – Two-dimensional example in augmented pattern space

5.7 - Adjusting the weight vector

We now suppose we are to train a single TLU with augmented weight vector w using the training set consisting of pairs like v, t. The TLU may have any number of inputs but we will represent that happening in pattern space in a schematic way using cartoon diagrams in 2D.

Suppose we present an input vector v to the TLU with desired response or target t = 1 and, with the current weight vector, it produces an output of y=0. The TLU has misclassified and we must make some adjustment to the weights. To produce a "0" the activation must have been negative when it should have been positive. Thus, the dot product w · v was negative and the two vectors were pointing away from each other as shown on the left hand side of figure 5.7.A.

In order to correct the situation we need to rotate w so that it points more in the direction of v. At the same time, we don't want to make too drastic a change as this might upset previous learning. We can achieve both goals by adding a fraction of v to w to produce a new weight vector w′, that is

$$w' = w + \alpha v$$

where $0 < \alpha < 1$, which is shown schematically on the right hand side of figure 5.7.B.

Figure 5.7.A – TLU misclassification 1-0

Suppose now, instead, that misclassification takes place with the target t = 0 but y = 1. This means the activation was positive when it should have been negative as shown on the left in figure 5.7.B. we now need to rotate w away from v, which may be effected by subtracting a fraction of v from w, that is

$$w' = w - \alpha v$$

as indicated on the left of figure 5.7.B.

We can combine as a single rule in the following way

$$w' = w + \alpha(t-y)v$$

This may be written in terms of the change in the weight vector $\Delta w = w' - w$ as follows:

$$\Delta w = \alpha(t-y)v$$

Since the algorithm specifies that we make no change to w if it correctly classifies its input, the convergence theorem also implies that, once w_0 has been found, it remains stable and no further changes are made to the weights. The convergence theorem was first proved by Rosenblatt (1962), while more recent versions may be found in Haykin (1994) and Minsky & Papert (1969).

One final point concerns the uniqueness of the solution. Suppose w_0 is a valid solution to the problem so that $w_0 \cdot x = 0$ defines a solution hyperplane. Multiplying both sides of this by a constant k preserves the equality and therefore defines the same hyperplane. We may absorb k into the weight vector so that, letting $w'_0 = kw_0$ $w_0 \cdot x = 0$. Thus, if w_0 is a solution, then so too is kw_0 for any k and this entire family of vectors defines the same solution hyperplane.

5.8 - The Perceptron

This is an enhancement of the TLU introduced by Rosenblatt (Rosenblatt 1962) and show in next Figure 5.8.A. It consists of a TLU whose inputs come from a set of preprocessing association units or simple A-units. The input pattern is supposed to be Boolean, that is a set of 1s and 0s, and the A-units can be assigned any arbitrary Boolean functionality but are fixed – they don't learn. The depiction of the input pattern as a grid carries the suggestion that the input may be derived from visual image for example. The rest of the node functions are just like TLU and may therefore be trained in exactly the same way. The TLU may be thought of as a special case of the perceptron with a trivial set of A-units, each consisting of a single direct connection to one of the inputs. Indeed, sometimes the term "perceptron" is used to mean what we have defined as a TLU. However, whereas a perceptron always performs a linear separation with respect to the output of its A-units, its function of the input space may not be linear separable if the A-units are non-trivial.

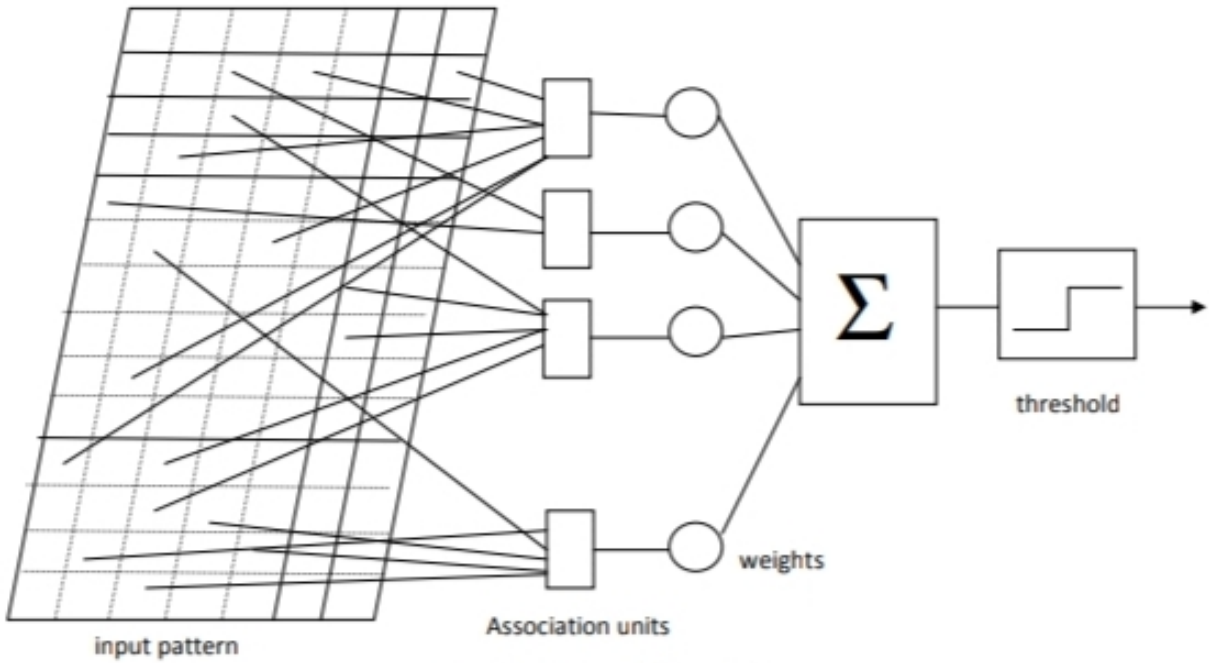


Figure 5.8.A – The Perceptron

5.9 - The Activation Functions

As mentioned previously, the activation function acts as a squashing function, such that the output of a neuron in a neural network is between certain values (usually 0 and 1, or -1 and 1). In general, there are several types of activation functions, denoted by Φ(.), some of them have been seen in Section 5.1, other common types are described follow in this Section.

There is the Threshold Function which takes on a value of 0 if the summed input is less than a certain threshold value (v), and the value 1 if the summed input is greater than or equal to the threshold value.

$$\varphi(v)=\begin{cases}1 & \text{if } v\geq 0\\ 0 & \text{if } v<0\end{cases}$$

Secondly, there is the Piecewise-Linear function. This function again can take on the values of 0 or 1, but can also take on values between that depending on the amplification factor in a certain region of linear operation.

$$\varphi(v)=\begin{cases}1 & v\geq \frac{1}{2}\\ v & -\frac{1}{2}>v>\frac{1}{2}\\ 0 & v\leq -\frac{1}{2}\end{cases}$$

Thirdly, there is the sigmoid function. This function can range between 0 and 1, but it is also sometimes useful to use the -1 to 1 range. An example of the sigmoid function is the hyperbolic tangent function.

$$\varphi(v)=\tanh\left(\frac{v}{2}\right)=\frac{1-\exp(-v)}{1+\exp(-v)}$$

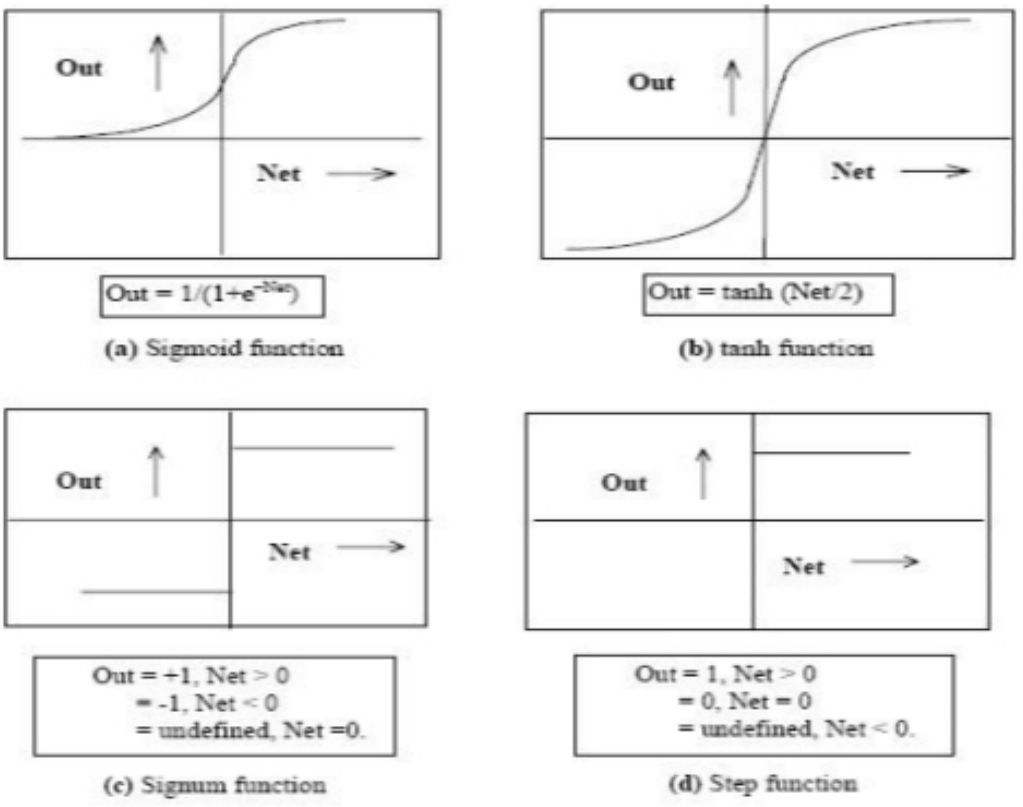


Figure 5.9.A – Common non-linear functions used as activation function. (a) Sigmoid or Logistic and (b) tanh, (c) Signum and (d) step.

5.10 - Single Layer Nets

Using the perceptron training algorithm we are now in position to use a single perceptron or TLU to classify two linearly separable classes A and B. Although the patterns may have many inputs, we may illustrate the pattern space in a schematic or cartoon way as shown in Figure 5.10.A. Thus the two axes are not labeled, since they do not correspond to specific vector components, but are merely indicative that we are thinking of the vectors in their pattern space.

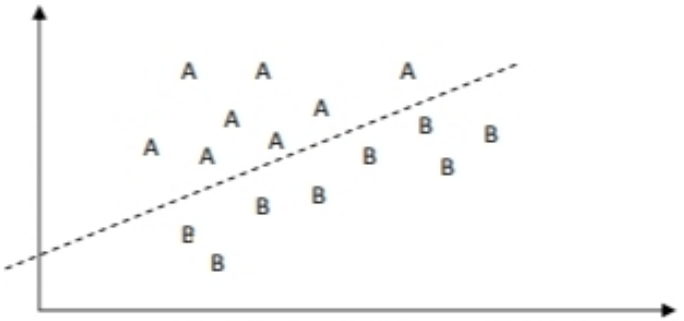


Figure 5.10.A – Classification of two classes A, B.

It is possible, however, to train multiple nodes on the input space to achieve set linearly separable dichotomies of the type shown in the figure 5.10.A. This might occur, for example, if we wish to classify handwritten alphabetic characters where 26 dichotomies are required, each one separating one letter class from the rest of the alphabet, "A"s from non "A"s, "B"s from non "B"s, etc. The entire collection of nodes forms a single-layer net as shown in Figure 5.10.B. Of course, whether each of the above dichotomies is linearly separable is another question. If they are, then the perceptron rule may be applied successfully to each node individually.

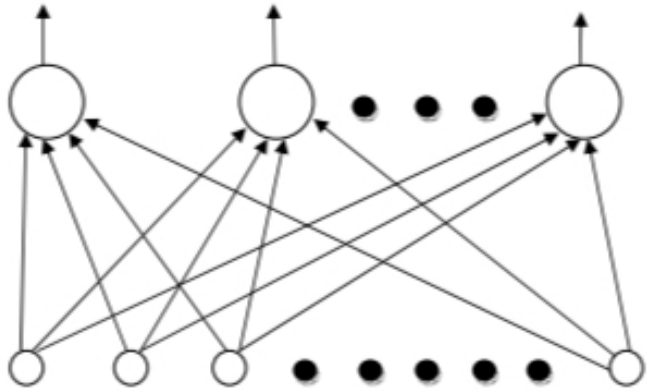


Figure 5.10.B – Single Layer Net

5.11 - How to choose the best Activation Function or TLU

Activation functions for the hidden units[1] are needed to introduce nonlinearity into the network. Without nonlinearity, hidden units would not make nets more powerful than just plain perceptrons (which do not have any hidden units, just input and output units). The reason is that a linear function of linear functions is again a linear function. However, it is the nonlinearity (i.e, the capability to represent nonlinear functions) that makes multilayer networks so powerful. Almost any nonlinear function does the job, except for polynomials. For backpropagation learning, the activation function must be differentiable, and it helps if the function is bounded; the sigmoidal functions such as logistic and tanh and the Gaussian function are the most common choices. Functions such as tanh or arctan that produce both positive and negative values tend to yield faster training than functions that produce only positive values such as logistic, because of better numerical conditioning (see ftp://ftp.sas.com/pub/neural/illcond/illcond.html).

[1] **HIDDEN UNITS**: in a neural network that mediate the propagation of activity between input and output layers. The activations or target values of such units are not specified by the environment, but instead arise from the application of a learning procedure that sets the connection weights into and out of the unit.

For hidden units, sigmoid activation functions are usually preferable to threshold activation functions. Networks with threshold units are difficult to train because the error function is stepwise constant, hence the gradient either does not exist or is zero, making it impossible to use backprop or more efficient gradient-based training methods. Even for training methods that do not use gradients--such as simulated annealing and genetic algorithms--sigmoid units are easier to train than threshold units. With sigmoid units, a small change in the weights will usually produce a change in the outputs, which makes it possible to tell whether that change in the weights is good or bad. With threshold units, a small change in the weights will often produce no change in the outputs.

For the output units, we should choose an activation function suited to the distribution of the target values:

- For binary (0/1) targets, the logistic function is an excellent choice (Jordan, 1995).
- For categorical targets using 1-of-C coding, the softmax activation function is the logical extension of the logistic function.
- For continuous-valued targets with a bounded range, the logistic and tanh functions can be used, provided you either scale the outputs to the range of the targets or scale the targets to the range of the output activation function ("scaling" means multiplying by and adding appropriate constants).
- If the target values are positive but have no known upper bound, we can use an exponential output activation function, but beware of overflow.
- For continuous-valued targets with no known bounds, use the identity or "linear" activation function (which amounts to no activation function) unless we have a very good reason to do otherwise.

There are certain natural associations between output activation functions and various noise distributions which have been studied by statisticians in the context of generalized linear models. The output activation function is the inverse of what statisticians call the "link function". McCullagh, P. and Nelder, J.A. (1989)

5.12 - Artificial Neural Networks – Real World Applications

The tasks to which artificial neural networks are applied tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.
- Robotics, including directing manipulators, computer numerical control.

Application areas include system identification and control (vehicle control, process control), quantum chemistry, game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (automated trading systems), data mining (or knowledge discovery in databases, "KDD").

5.13 - BPMaster Neural Network Simulator

Neural network simulators are software applications that are used to simulate the behavior of artificial or biological neural networks. They focus on one or a limited number of specific types of neural networks. They are typically stand-alone and not intended to produce general neural networks that can be integrated in other software. Simulators usually have some form of built-in visualization to monitor the training process. Some simulators also visualize the physical structure of the neural network.

BPMaster is a software developed in the Stanford University, and was created by J. L. McClelland and D. E. Rumelhart's Explorations in Parallel Distributed Processing: A handbook of models, programs, and exercises (Cambridge, MA: MIT Press).

After that, in the Universitat Politècnica de Catalunya, was updated with new functionalities like set/ ensayo, "ensayo" or "backsel" by Enrique Romero and J.M. Sopena.

We have used the "ensayo" test, which make three data groups (train, validation and test), using a logistic activation function.

Using the program BPmaster we have developed the neural network training experiment, using the logistic activation function in a simple neural network. The program BPmaster has a command called "ensayo" that allows this type of neural training with options in a number of activation functions.

Once executed the training's the test results are presented as train, validation and test, those are the averaged of the results of each epoch made, from the validation results, we obtain the average neural training results.

At the same time cross-validation is performed, allowing to know the accuracy on the validation test set.

Cross validation is the practice of partitioning the available data into multiple subsamples such that one or more of the subsamples is used to fit (train) a neural model, with the remaining subsamples being used to test how well the model performs. Because the model is tested on data not used in training the model, cross validation can provide a useful estimate of how the model will perform on new data.

The number of cross-validations is the number of times that the whole process of cross-validation is run. Therefore each cross-validation the order of the patterns changes.

You can type in the output file the:

- Results of each epoch
- Summary of results of each test
- Summary of the set of all tests

The results labeled training, validation and test are interpreted as:

1. Training: Results (in training, validation and test sets) using the network giving the optimal training results.
2. Validation: Results (in training, validation and test sets) using the network giving the optimal validation results.
3. Test: Results (in training, validation and test sets) using the network giving the optimal test results.

The first is to see how much overfitting exists. The second is what gives the result of generalization (test results in the optimal validation). The third is an estimate of the best that could have been obtained.
In the end, the means of the results of each test are displayed (this is where we have to look at the result of generalization).

6 - Feature Classification and Selection

Building accurate predictive models in many domains requires consideration of hundreds of thousands or millions of features. Models which use the entire set of features will almost certainly over fit on future datasets. Standard statistical and machine learning methods such as SVMs, maximum entropy methods, decision trees and neural networks generally assume that all features are known in advance. They use regularization (e.g. ridge regression, weight decay in neural networks or Gaussian) or feature selection to avoid over fit.

Feature selection offer many benefits beyond improved prediction accuracy. For example, feature selection can reduce the requirements of measuring and storing data. Non- predictive or redundant expensive features may not need to be collected or captured. Feature selection can also speed up model updating. When large amount of new information are available on a minute by minute basis, the time to update models can be prohibited if large sets of features are involved. Selecting the most important features from the new information will speed up the model updating. Other needs include better data understanding.

This section is focused on selecting a subset of features for spam classification attributes to build a good predictive model for ANN antispam.

6.1 - Feature Selection brief review and definitions

The feature selection model typically assumes a setting consisting of *n* observations and a fixed number *m* of candidate features.

The goal is to select the feature subset that will ultimately lead to the best performing predictive model. The size of the search space is therefore 2^m, and identifying the best subset is NP-complete. Many commercial statistical packages offer variants of a greedy method, "forward feature selection", an iterative procedure in which at each step all features are tested and the single best feature is selected and added to the model. Feature Selection performs hill climbing in the space of feature subsets. Feature selection is terminated when either all candidate features have been added, or none of the remaining features lead to increased expected benefit according to some measure, such as a p-value threshold. Variants of stepwise selection abound, including forward (adding features deemed helpful), backward (removing features no longer deemed helpful), and mixed methods (alternating between forward and backward). Our study and work will assume a simple forward search.

There are many methods for assessing the benefit of adding a feature. Computer scientists tend to use cross-validation, where the data set is divided into several (say k) batches with equal sizes. K-1 of the batches are used for training while the remainder batch is used for evaluation. The training procedure is run k times so that the model is evaluated once on each of the batches and performance is averaged. The approach is computationally expensive, requiring k separate training steps for each evaluation. A second disadvantage is that when observations are scarce the method does not make good use of the observations.

A fundamental problem of machine learning is to approximate the functional relationship $f(\ )$ between an input $X = \{x1, x2, \ldots, xM\}$ and an output Y, based on a memory of data points, $\{Xi, Yi\}$, $i = 1, \ldots, N$, usually the Xi's are vectors of reals and the Yi's are real numbers. Sometimes the output Y is not determined by the complete set of the input features $\{x1, x2, \ldots, xM\}$, instead, it is decided only by a subset of them $\{x(1), x(2), \ldots, x(m)\}$, where m < M . With sufficient data and time, it is fine to use all the input features, including those irrelevant features, to approximate the underlying function between the input and the output. But in practice, there are two problems which may be evoked by the irrelevant features involved in the learning process.

The irrelevant input features will induce greater computational cost. For example, using cached *k*d-trees, locally weighted linear regression's computational expense is $O(m^3 + m^2 \log N)$ for doing a single prediction, where N is the number of data points in memory and m is the number of features used. Apparently, with more features, the computational cost for predictions will increase polynomially; especially when there are a large number of such predictions, the computational cost will increase immensely.

The irrelevant input features may lead to overfitting. For example, in the domain of medical diagnosis, our purpose is to infer the relationship between the symptoms and their corresponding diagnosis. If by mistake we include the patient ID number as one input feature, an over-tuned machine learning process may come to the conclusion that the illness is determined by the ID number.

Another motivation for feature selection is that, since our goal is to approximate the underlying function between the input and the output, it is reasonable and important to ignore those input features with little effect on the output, so as to keep the size of the approximator model small.
For example, [Akaike, 73] proposed several versions of model selection criteria, which basically are the trade-offs between high accuracy and small model size.

The feature selection problem has been studied by the statistics and machine learning communities for many years. It has received more attention recently because of enthusiastic research in data mining. According to [John et al., 94]'s definition, [Kira et al, 92] [Almuallim et al., 91] [Moore et al, 94] [Skalak, 94] [Koller et al, 96] can be labelled as "filter" models, while [Caruana et al., 94] [Langley et al, 94]'s research is classified as "wrapped around" methods. In the statistics community, feature selection is also known as "subset selection", which is surveyed thoroughly in [Miller, 90].
The brute-force feature selection method is to exhaustively evaluate all possible combinations of the input features, and then find the best subset. Obviously, the exhaustive search's computational cost is prohibitively high, with considerable danger of overfitting. Hence, people resort to greedy methods, such as forward selection. In this work, we propose three greedier selection algorithms in order to further enhance the efficiency. We use real-world data sets from email world from different users to obtain the accuracy near the 95% of spam detection using Artificial Neural Networks.

Forward Feature Selection Algorithm

In this section, we introduce the conventional feature selection algorithm: forward feature selection algorithm; forward algorithm, in order to improve the computational efficiency without sacrificing too much accuracy.

The forward feature selection procedure begins by evaluating all feature subsets which consist of only one input attribute. In other words, we start by measuring each attribute error for every one-component subsets, $\{X1\}, \{X2\}, \ldots, \{XM\}$, where M is the input dimensionality; so that we can find the best individual feature, $X(1)$.

FS:= 0
Candidates:= {1,2,...m} // all features

For j := 1 to m do

For every feature K in candidate do

Extended FS:= FS U {K}

Evaluate extended FS using P-partitional dataset and store it in BestExtendedFS If it is the best among the subsets teste in this inner loop;

End For

Let K_{Best} be the feature added in BestExtendedFS;

FS:=BestExtendedFS; // FS:= U $\{K_{Best}\}$

Candidates := Candidates – $\{K_{Best}\}$;

Evaluate extended FS using P-partitional dataset and store it in BestFS If it is the best among the subsets teste in this inner loop;

End For

Return Best FS;

Algorithm 6.3.A – Feed Forward Algorithm for Feature Selection

Procedure evaluate FS using P-partitional Dataset

For each fold (i=0,1,2,.....P-1)

Let Trainset(i):= all folds except I;

Let Trainset(i):= the $i^{'th}$ fold;

Train an ANN_i with TrainSet(i) using FS;

$TestScore_i$:= RMS score of ANN_i onTestSet(i)

End For

Return the mean Test Score for the P values of TestScore

Algorithm 6.3.B – Cross-validation for P-partitional Dataset

Procedure evaluate FS using Double Cross-Validation and P-partitional Dataset

For each fold (i=0,1,2,.....P-1)

Let Testset(i):= the $i^{'th}$ fold;

For each fold j ≠ i

Let Validationset (i,j) := the $j^{'th}$ fold;

Let Trainset (I,j) := all folds except i and j;

Train an ANN_{ij} with Trainset(i,j) and Validationset(i,j) using FS;

$TestScore_{ij}$:= RMS error of ANN_{ij} on TestSet(i)

End For

End For

Return the mean TestScore for the P*(P-1) values of $TestScore_{ij}$

Algorithm 6.3.C – Double Cross-validation for P-partitional Dataset

Where the computational cost in single-layer perceptron will be:

$$\text{Computational Cost} = O(P^2 * \text{MaxEpoch} * N * m)$$

The goal of feature selection is to choose a subset X_S of the complete set of input features $X = \{x_1, x_2, \dots, x_M\}$ so that the subset X_S can predict the output Y with accuracy comparable or even better to the performance of the complete input set X, and with great reduction of the computational cost.

First, let clarify how to evaluate the performance of a set of input features. Even if feature sets are evaluated by testset cross-validation or leave-one-out cross validation, an exhaustive search of possible feature-sets is likely to find a misleadingly well-scoring feature-set by chance. To prevent this, we use the cascaded cross-validation procedure in Figure N upper, which selects from increasingly large sets of features (and thus from increasingly large model classes).

To evaluate the performance of a feature selection algorithm is more complicated than to evaluate a feature set. This is because in order to evaluate an algorithm, we must first ask the algorithm to find the best feature subset. Second, to give a fair estimate of how well the feature selection algorithm performs, we should try the first step on different datasets.

6.2 - Feature selection algorithms

Now we will introduce the conventional feature selection algorithm: forward feature selection algorithm; then we explores briefly the variants of the forward algorithm for study porpoise, and briefly we will describe each feature selection variant.

6.2.1 - Exhaustive Search feature selection

To find the overall best input feature set, we can also employ exhaustive search. Exhaustive search begins with searching the best one-component subset of the input features, which is the same in the forward selection algorithm; then it goes to find the best two-component feature subset which may consist of any pairs of the input features. Afterwards, it moves to find the best triple out of all the combinations of any three input features, etc. It is straightforward to see that the cost of exhaustive search is the following:

$$\text{MC}(1) + \binom{\text{M}}{2} C(2) + \cdots + \binom{\text{M}}{\text{m}} \text{C(m)}$$

Compared with the exhaustive search, forward selection is much cheaper. Exhaustive search has the most expensive computational cost.

6.2.2 - Backward feature selection "backsel"

Backward elimination or Backward feature selection is a feature search starting with the full feature set. At each iteration the next candidate subsets are formed by eliminating each feature, one at a time, from the current subset. The feature whose elimination resulted in the highest performance improvement is eliminated permanently. The search stops when the termination condition is met or only one feature remains in the set.
Backward feature selection has very high computational cost.

The computational cost is:

BS = 1 C(computational cost(1) + 2 C(computational cost(2)+.... + m C(computational cost(m)

Although the cost is asymptotically similar to the cost of forward feature selection, is not the same computationally, due to not so efficiency compared to Forward Feature Selection, as we will see in the following section.

6.2.3 - Forward feature selection

The forward feature selection procedure begins by evaluating all feature subsets which consist of only one input attribute. In other words, we start by measuring the Cross-Validation (CV) error of the one-component subsets, $\{X_1\}$, $\{X_2\}$, ..., $\{X_M\}$, where M is the input dimensionality; so that we can find the best individual feature, $X_{(1)}$.
Next, forward selection finds the best subset consisting of two components, $X_{(1)}$ and one other feature from the remaining M-1 input attributes. Hence, there are a total of M-1 pairs. Let's assume $X_{(2)}$ is the other attribute in the best pair besides $X_{(1)}$.

Afterwards, the input subsets with three, four, and more features are evaluated. According to forward selection, the best subset with m features is the m-tuple consisting of $X_{(1)}, X_{(2)}, ..., X_{(m)}$, while overall the best feature set is the winner out of all the M steps. Assuming the cost of a CV evaluation with i features is C(i), then the computational cost of forward selection searching for a feature subset of size m out of M total input attributes will be

$$MC(1) + (M-1)C(2) + ... + (M-m+1)C(m)$$

For example, the cost of one prediction with ANN using Single-Layer Perceptron will be:

Computational Cost = O (M^2 * C(evaluation cost FS))

Or

Computational Cost = O (M^3 * P^2 * Max_{epoch} * N)

Where M^3 is the total of candidates and P^2 is the P partitioned dataset and M(epoch) the

Maximum quantity of epoch used by the ANN and N the number of datapoints.

In others words

FS = m C(computational cost(1) + m-1 C(computational cost(2)+.... + 1 C(computational cost(m)

However, forward selection may suffer because of its greediness. For example, if $X_{(1)}$ is the best individual feature, it does not guarantee that either $\{X_{(1)}, X_{(2)}\}$ or $\{X_{(1)}, X_{(3)}\}$ must be better than $\{X_{(2)}, X_{(3)}\}$. Therefore, a forward selection algorithm may select a feature set different from that selected by exhaustive searching. With a bad selection of the input features, the prediction Y_q of a query $X_q = \{x_1, x_2, \ldots, x_M\}$ may be significantly different from the true Y_q.

6.2.4 - Restricted Forward Selection (RFS)

1- Calculate all the 1-feature set using any classification techniques to obtain the errors, and sort the features according to the corresponding errors. Suppose the features ranking from the most important to the least important are $X_{(1)}, X_{(2)}, \ldots, X_{(M)}$.

2- Do the classification techniques of 2-feature subsets which consist of the winner of the first round, $X_{(1)}$, along with another feature, either $X_{(2)}$, or $X_{(3)}$, or any other one until $X_{(M/2)}$. There are M / 2 of these pairs. The winner of this round will be the best 2-component feature subset chosen by RFS.

3- Calculate using the classification techniques to obtain errors of M / 3 subsets which consist of the winner of the second round, along with the other M / 3 features at the top of the remaining rank. In this way, RFS will select its best feature triple.

4- Continue this procedure, until RFS has found the best m-component feature set.

5- From Step 1 to Step 4, RFS has found m feature sets whose sizes range from 1 to m. By comparing their errors, RFS can find the best overall feature set.

The difference between RFS and conventional Forward Selection (FS) is that at each step to insert an additional feature into the subset, FS considers all the remaining features, while RFS only tries a part of them which seem more promising.

6.2.5 - Greedy Algorithm

Do all the 1-attribute using any classification techniques and sort them, take the best two individual features and evaluate their error, then take the best three individual features, and so on, until m features have been evaluated. Compared with the super greedy algorithm, this algorithm may conclude at a subset whose size is smaller than m but whose inner testset error is smaller than that of the m-component feature set. Hence, the greedy algorithm may end up with a better feature set than the super-greedy one does.

6.2.6 - Super Greedy Algorithm

Do all the 1-attribute using any classification techniques calculations, sort the individual features according to their mean error, then take the m best features as the selected subset. We thus do M computations involving one feature and one computation involving m features.

After studying all forms of feature selection described in the previous sections, now we conducted the experiments using the feed forward feature selection algorithm with Single-Layer Perceptron Artificial Neural Network.

This algorithm delivers a level of acceptable compromise between the computational cost and quality of search features. As we have seen the rest of algorithms such as Backward has a very high computational cost, or as the Greedy algorithms that although it's computational cost is very low, the accuracy of the features it provides is highly questionable.

The forward selection used in this work has been the Forward Feature Selection. We resort to experiments using real world spam dataset for spam and ham classification. You may see their results in the following chapter.

7 - Dataset Creation "the email corpus"

In this project we have used a corpus of 2200 e-mails received from different senders to different receivers collected by the ISP (www.masbytes.com in Spain) over a period of several months. Some of the e-mails contained embedded attachments.

The corpus was hand (visual-human review) checked to classify spam and ham, follow-on 1100 ham and 1100 spam.

Each e-mail message was saved as a text file or HTML, and then parsed to identify each header element (such as Received: or Subject:) to distinguish them from the body of the message.

We have taken several attributes or features from those emails, every attribute has been obtained based in our expertise looking how spammers are making the email undetectable, then we have defined some features for in spam recognition(using for example, antivirus, char, vowels, etc).
The objective was to improve the feature selection using not only email attribute but also characterization using opensource software as SPF, ClamAV, Whitelist, blacklist, etc. You may see more references in the first section of this work.

7.1 – How we have obtained "the corpus"

Due to the large amount of anti-spam filters available today, results of performance testing have been studied separately within the open source community to verify the validity of each one of them.

Therefore the filters implemented and executed, have a high probability to be very valid and secure when detect both spam and ham.

Here you will find a detailed description of the filters used. For more information, please visit sourceforge.net.

Charts blots: Unlike other antispam systems, lists may apply SINESPAM blots of IP before any other filter. This ensures that the client can receive mail from specific servers but these servers are labeled as disreputable.

Ips Reputation: The second layer of filtering is the reputation of IP and RBLs. It checks the reputation of the source server so that by studying their behavior, both historical and current, you can categorize your mail and eventually eliminate up to 90% of spam. Not only was immensely successful in reducing the amount of spam but it makes the most efficient way to close the connection to the spammer even before receiving the mail. In these cases, the spammer detects that you are not accepted mail and takes it into account when deciding on the least protected domains to which address the following attacks. To get the number of false positives is virtually zero SINESPAM leaving no mail is not at least in two of the six RBLs consulted.

Charts blots by domain or email address: Both the manager and the user can enter addresses and domains known to be sure they will not be filtered by creating false positives.

Lists of trust: trust lists are automatically with valid mailing addresses that regularly receives each user. This list is personal and is generated automatically, using a proprietary algorithm to SINESPAM that ensures the reliability of the accounts. Thanks to the trust lists avoiding false positives without the user having to intervene at any time.

SPF: Sender Policy Framework Using is achieved to ensure that the servers from which SINESPAM receives the mail are allowed to send mail from a specific domain. This technique prevents email spoofing, or phishing. In order to use SPF, it must be properly configured on the source servers.

Razor Servers: spam-catcher using a collaborative filtering network to Vipul's Razor is in distributed, collaborative, spam detection and filtering network. Establishes a Razor in distributed and Constantly updating catalog of spam in propagation. This catalog is Used by clients to filter out spam Known. On Receiving a spam, a Razor Reporting Agent (run by an end-user or a troll box) and submits Calculates a 20-character unique identification of the spam (a SHA Digest) to ITS Closest Razor Catalogue Server.

SenderDomval: It verifies the existence of the domain of the sender to remove the spam that is sent from nonexistent domains.
Sendercallout: It verifies the existence of the sender to delete the spam that is sent from nonexistent accounts.

Greylisting: The emails are categorized according to the probability that they are valid. When the score they receive does not ensure that the emails are valid, they can apply greylisting technique of allowing a temporary error to the sending server. If the server is sending spam, not normally retried, while if the mail is valid, the server must (if properly configured) to retry sending after some time.

Delay: The emails are categorized according to the probability that they are valid. When the score they receive does not ensure that the emails are valid, you can apply a certain delay in the connection that criminalizes the sending server. If it is a spam server does not want to waste time and cut the connection to try other servers.

Content Filter: A way to customize the filtering is to allow the administrator and individual users the ability to create their own filters. These filters can be created by a company or user and work analyzing the words mail and acting according to its settings.

Bogofilter and Spamassassin: Two of the modules that are used in SINESPAM Bogofilter and Spamassassin. These are based on different antispam techniques ranging from Bayesian filters to DNS-based tests or consultations in Databases. The rules and evidence contained in these two systems are constantly adjusted for optimum performance depending on the type of spam. Some of the tests performed are:

> **Inspection of "Headers":** The "Headers" or message headers contain important information about it.
> **Analysis of the Message**: The body of the message and title are read by SpamAssassin, by searching by keywords or structures that make up a spam email.
> **Analysis probabilistic / Bayesian**: Once you've set the initial rules for the detection, probabilistic analysis was performed to determine similarities between incoming messages and those already previously identified as SPAM.

Lists "Hash" Signatures Mail: Because a SPAM email usually sent thousands of people at once, the structure of each message is identical in all instances, thus producing a "Hash" unequivocal. SpamAssasisn query lists of "hashes" on messages known.

Antivirus: Virus scanning is applied to all emails within the system, whether valid or is considered spam. The virus is constantly updated automatically. There is the possibility to disable anti-virus filtering from the control panel. (SpamCop and CLAM-AV)

The definition of the 31 attributes initially proposed is given in the following table.

7.2 - Features from the Message Body and Header

Attributes definition for the emails collected, to use in the learning system based on Artificial Neural Nets.

We have two types of components within the email, the first is the header (which defines the SMTP protocol headers) and the second body of the email (where email drafted Itself).

The following table shows that to detect patterns in email spam, you must search or inspect both the email header as in the body.

The Table is divided into two groups header and body, in the first column is the description of the attribute to be inspected in the next column is the value of the variable definition for the email corpus and the third column definition the variable itself.

NOTE: Some of the attributes described in the following table not have been used in extracting the information contained the values in themselves, because they had no certain and quantifiable results.

Feature Definition	Value	Variable
HEADER		
Attribute Spamassasin – Verify email has passed spamassasin spam= 1 nospam=0	Binary YES=1 NO=0	att_spamassasin
Sender Blacklisted rbl=1 norbl=0	Binario YES=1 NO=0	att_rbl
Attribute char extended – char extended quantity: ¡ , ¢ , £ , ¤ , ¥ , ¦ , § , ¨ , © , ª , « , ¬ , ® , ¯ , ° , ± , ² , ³ , ´ , µ , ¶ , · , ¸ , ¹ , º , » , ¼ , ½ , ¾ , ¿ , À , Á , Â , Ã , Ä , Å , Æ , Ç , È , É , Ê , Ë , Ì , Í , Î , Ï , Ð , Ñ , Ò , Ó , Ô , Õ , Ö , × , Ø , Ù , Ú , Û , Ü , Ý , Þ , ß , à , á , â , ã , ä , å , æ , ç , è , é , ê , ë , ì , í , î , ï , ð , ñ , ò , ó , ô , õ , ö , ÷ , ø , ù , ú , û , ü , ý , þ , ÿ	Number = Char Quantity	att_char_extend
Verify the name sender has only alphabetic words yes=1 or no= 0	Binary YES=1 NO=0	Not corresponding bad feature
Verify the name sender has only vowels yes = 1 or no= 0	Binary YES=1 NO=0	Not corresponding bad feature
Number of alphabetic words that did not contain any vowels on Header	Number = Char Quantity	att_char_header_consonante
Number of alphabetic words that contained at least two of the following letters (upper or lower case): K, Q, X, Z, Y, W,V on header	Number = Char Quantity	att_char_nospanish

Number of alphabetic words that were at least 15 characters long – Not URL	Number = Char Quantity	att_long_words
Number of words with all alphabetic characters in upper case	Number = Char Quantity	att_char_alluppercase
Verify SPF Sender tiene spf=1 nospf=0	Binary YES=1 NO=0	att_spf
Binary feature indicating whether a priority header appeared within the message headers (X-Priority and/or X-MSMail-priority) or whether the priority had been set to any level besides normal or medium: yes = 1, no = 0	Binary YES=1 NO=0	att_header_priority
Binary feature indicating whether a content-type header appeared within the message headers or whether the content type of the message has been set to "text/html": yes = 1, no = 0 Features From the Message Body	Binary YES=1 NO=0	att_header_html
Verify the Sender from valid MX -	Binariy YES=1 NO=0	att_mx_sender
Verify Virus email – passed by the clamAV antivirus	Binary YES=1 NO=0	att_virus
Verify email has passed OK by the sanesecurity spam database spam=1 nospam=0	Binary YES=1 NO=0	att_sane_spam
Verify email has passed OK by the sanesecurity virus database virus=1 novirus=0	Binary YES=1 NO=0	att_sane_virus
Verify RAZOR Sender yes=1 no=0	Binary YES=1 NO=0	att_razor
Verify PYZOR Sender yes=1 no=0	Binary YES=1 NO=0	att_pyzor
Verify DCC Sender dcc=1 nodcc=0	Binary YES=1 NO=0	att_dcc
BODY		
Number of URLs within hyperlinks that contain any numeric digits or any of three special characters ("&", "%" or "@") in the domain or subdomain(s) of the link	Number = URLs Hiperlink Quantity	att_body_href_extend_char
Verify the message header subject has been signed by a real sender. – DKIM	Binary YES=1 NO=0	att_dkim
Binary feature indicating occurrence of a character (including spaces) that is repeated at least three times in succession: yes = 1, no = 0 Features From the Priority and Content-Type Headers	Binary YES=1 NO=0	att_char_repeat
Char word extended inside word like V1agr3 – s1agra – character number or extended character inside word without spaces	Number = Char Quantity	att_char3_number
Attribute Spanish – How many spanish letters has the email: á , é , í , ó , ú , Á, É , Í , Ó, Ú, ñ , Ñ	Number = Char Quantity	att_spanish
Verify the email has passed clean by the CRM 114 antispam system with dictionary	Binary YES=1 NO=0	att_crm114
The run-length attributes (55-57) measure the length of sequences of consecutive capital letters - CIC Not URL Not emails	Number = Char Quantity	Not corresponding bad feature