Sistemas Inteligentes: Reportes Finales Ago-Dic 2013

Book · December 2013

2 authors, including:

Gildardo Sanchez-Ante
Universidad Politécnica de Yucatán, México
40 PUBLICATIONS **810** CITATIONS

..... Reporte Tecnico .....

Sistemas Inteligentes: Reportes Finales Ago-Dic 2013

Gildardo Sanchez-Ante, Editor
Tecnológico de Monterrey
Campus Guadalajara

in: Reporte Tecnico RT-0002-2013. See also BIBTEX entry below.

BIBTEX:

```
@article{Sistemas Inteligentes: Reportes Finales Ago-Dic 2013,
  author      = {Gildardo Sanchez-Ante, Editor},
  title       = {Sistemas Inteligentes: Reportes Finales Ago-Dic 2013},
  journal      = {Reporte Tecnico RT-0002-2013},
  institution = {Tecnologico de Monterrey, Campus Guadalajara},
  month       = {Diciembre},
  year        = {2013},
}
```


document created on: December 14, 2013
created from file: editor.tex
cover page automatically created with CoverPage.sty
(available at your favourite CTAN mirror)

TECNOLÓGICO DE MONTERREY®

Preface

Estos reportes normalmente se generaban cada año, en virtud de que el curso de Sistemas Inteligentes se programaba en el semestre Enero-Mayo. Sin embargo, este año el curso se ofreció tanto en Enero como en Agosto, razón por la cual tenemos dos reportes en 2013. En esta ocasión hubo mas proyectos, pues los estudiantes trabajaron individualmente en su mayoría.

En cuanto a la temática de los proyectos, este semestre tuvimos varios equipos interesados en temas comunes, así que los he agrupado acorde con ello en el reporte. Un grupo por ejemplo, se interesó en analizar el problema de reconocimiento de señales de tránsito, y en virtud de que existen varias bases de datos con imágenes de acceso público, resultaba factible trabajar en ese problema. Asi, hubo quienes probaron diversos métodos de clasificación. En algunos casos, el enfoque ha sido la determinación de los valores mas adecuados para los parámetros del método en cuestión. En otros casos se ha trabajado con la determinación de un buen conjunto de rasgos descriptores.

Otro grupo de estudiantes se interesó por probar soluciones al problema de clasificación de correos electrónicos como spam. La mayoría de los métodos probados caen en el ámbito de el razonamiento probabilista.

Asi mismo, hubo varios estudiantes que trabajaron con estrategias para juegos, encontrando aquí desde un juego de Pokemon, hasta Tetris, con enfoques muy interesantes.

Finalmente, de manera ya mas puntual, otros trabajos se enfocaron a aplicaciones de medicina, finanzas, optimizacin y seguridad computacional.

Asi pues, el lector encontrará aquí una buena variedad de ideas que se han explorado en este semestre. En esta ocasión mi labor de edición ha sido menos intrusiva, por lo que los artículos son prácticamente los que los estudiantes entregaron.

Guadalajara, Jalisco, *Gildardo Sánchez-Ante*
Diciembre, 2013

Contents

Chapter 1
Traffic Sign Recognition

Traffic Sign Recognition using Histograms of Oriented Gradients and Artificial Neural Networks

Adrian Garcia Betancourt

Abstract In this paper, an approach to classification of traffic sign images is studied. Images are divided into four categories, each corresponding to one traffic sign. HOG descriptors are used in conjunction with an implementation of a neural network to perform the task. As shown by the experiment results, the proposed method is effective, but not as much as needed by today's cutting-edge technologies.

1 Introduction

Traffic signs are designed to be easy to read for humans, but for computer systems, the task of classifying traffic signs poses a challenging pattern recognition problem. Efforts on image processing and machine learning algorithms have been made to improve this task, although little comparison of such approaches exists (Stallkamp, Schlipsing, Salmen, & Igel). Traffic sign recognition is an important subject as it relates to new technologies, such as drivers assisting systems, and automated navigation. It involves different technologies; pattern recognition, digital image processing, artificial intelligence, and computer vision are examples of these technologies (Xiaoguang, Xinyan, Deren, & Hui).

It is the objective of this study to give an approach for the task of classifying traffic signs. For the purpose of achieving this objective, four traffic sign classes were selected among the forty-three different classes available in the German Traffic Sign Recognition Benchmark dataset (GTSRB). More on the GTSRB follows in the next section. The four classes were selected based on the visual differences between them. They correspond to the four pitcures shown in Figure 1. The approach taken for the accomplishment of the task was to build a computer program, which would do the task of classifying from a given set of test data, into the four classes described in the image above.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201. e-mail: a01087680@itesm.mx

More specifically, in the first stage, the computer program iterates through a given set of images, and extracts one corresponding feature vector for each image it scans; all of the feature vectors are written into a text file, from which the program will read the training data, in order to train for the classification task. In the second stage, an artificial neural network is built, and it is trained with the training data text file; as a result, the newly created neural network is written into another text file. Finally, in the third stage, the program recreates the neural network from the text file, and uses it to perform the classification task among a set of images given in the test image set. This 3-phase approach is better illustrated in Figure 2.

In the following sections, an introductory background is given on each subject involved in the approach taken; and more details and explanations are given on the steps taken to accomplish the main goal.

2 Used Technologies

The next subsections list the technologies applied for the accomplishment of the objective of this study. An introductory background is given for each topic so that the reader can understand the steps taken in the proposed approach to traffic sign classification.

Fig. 1: The four traffic signs chosen to work with. From left to right, the images relate to class numbers 11, 14, 33, and 44 from the original GTSRB dataset

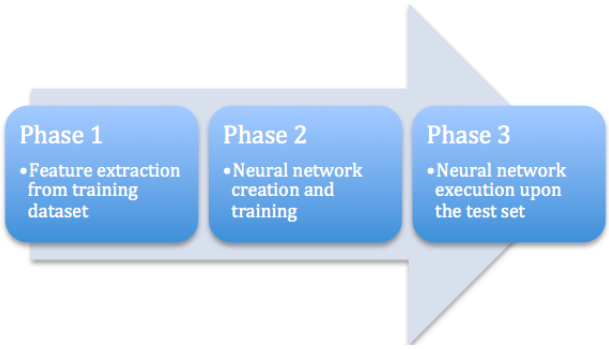


Fig. 2: The 3-phase approach for the accomplishment of the task

2.1 German Traffic Sign Recognition Benchmark

The German Traffic Sign Recognition Benchmark (GTSRB) was a competition held at IJCNN in the year 2011. It consisted of a multi-category classification problem with unbalanced class frequencies; the goal being to classify a vast image repository into forty-three different traffic sign classes. A website with a public leaderboard was set up and will be permanently available for submission of new results, among with the availability of the training and testing datasets. This study is based on subsets of the training and testing datasets available at the GTSRB website.

The dataset at GTSRB was created from more or less ten hours of video recording while driving along different roads in Germany. Data collection, annotation and image extraction was performed afterwards. The set contains images of approximately 1,700 traffic sign instances (real-world traffic sign objects). The size of the images is between 15×15 and 222×193 pixels. These images contain a 10% wide margin to allow for the usage of edge detectors. The extended dataset consists of 51,840 images of German road signs among 43 different classes (Stallkamp, Schlipsing, Salmen, & Igel).

2.2 Histograms of Oriented Gradients

The Histogram of Oriented Gradients (HOG) image feature extraction method consists on evaluating well-normalized local histograms of image gradient orientations in a dense grid. Local object appearance and shape can be represented rather well by the distribution of edge directions or local intensity gradients. The algorithm divides the image window into small regions called cells, while each cell accumulates a local histogram of edge orientations over the pixels of itself. The combined histogram entries form the representation of the image.

In order to provide for better invariance to illumination, the HOG algorithms use larger spatial regions called blocks, which consist each of several cells. The blocks accumulate a measure of local histogram energy and use the results to normalize all of the cells within the block. Thus, the HOG algorithm receives an image object as an input, and via computing of gradients and normalization of colour, provides a set of HOG descriptors as output. The HOG descriptors can then be merged into a single vector which will correspond to the feature representation of the original image (Dalal & Triggs, 2005).

2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) are massively parallel computing systems that consist of a very large number of simple processors with many interconnections. ANN models try to use some principles believed to be used in the human brain. A

biological neuron is an information processing cell composed of cell body, called the soma, and many tree-like branches, classified into axon or dendrites. A biological neuron receives impulses from its neighbour neurons through its dendrites (receivers or inputs), and transmits signals generated by itself along the axon (transmitter or output).

A synapse is a functional unit between two neurons. It consists of certain chemicals, called neurotransmitters, being passed from one neurons output to a neighbour neurons input. The neurotransmitters are capable of enhance or inhibit the receptors neuron own tendency to emit signals, depending on the type of the synapse. The cerebral cortex of humans contains approximately 10^{11} neurons; they are massively connected, totalling between 10^{14} to 10^{15} connections (Anil, Jianchang, & Mohiuddin). A representation of a biological neuron is shown in Figure 3.

Based on the biological model of neurons, there have been several mathematical models of neurons, called artificial neurons, which resemble the behaviour of biological neurons but within a mathematical context appropriate for problem solving. One example is McCulloch and Pitts model of a mathematical neuron, found in A logical calculus of ideas immanent in nervous activity. This model of neuron consists of a binary threshold unit. It computes a sum of its n input signals, giving a corresponding weight to each one, and generates an output of one or cero, depending on a specified threshold.

Based on the models of the artificial neuron, an artificial neural network can be seen as a computer program that operates analogous to the neural network present in the brain. A distinguishing feature of ANNs is that the knowledge is not explicitly written into the program, but rather distributed throughout the network. ANNs are built processing elements (artificial neurons) grouped into layers; the input layer of neurons receives information, and the output layer propagates the response. Between the input and output layers, there may be several hidden layers, where most part of the information processing takes place. The output of the network depends on the individual weights given to each connection.

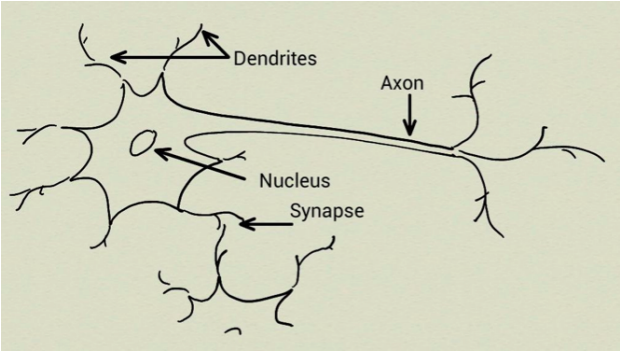


Fig. 3: Model of a biological neuron

Training an artificial neural network involves exposing the network to examples where the input data and the desired output data is specified. After the network has been trained, it can be exposed to new instances of the data and used to classify new cases (Zwass).

2.4 Dlib C++ Library

Dlib is an open source, general purpose, cross-platform C++ library. It provides several modules, which deal with different problem-domain areas. One of the modules includes support for image-processing tasks, such as image I/O and feature extraction (Major Features). For the purpose of this study, the Dlib library was used for the tasks of image input and feature extraction. In concrete, the Dlib image processing classes deal with images based on 2-dimensional arrays, called `array2d`, in which pixels can be stored to represent the original image. The types of pixels that this representation support are `rgb_pixel`, `bgr_pixel`, `rgb_alpha_pixel`, and `his_pixel`.

The feature extractions algorithms supported by Dlib include C++ classes and routines that automatically handle this task. Among them, we find `hog_image` and `extract_fhog_features`. The object `hog_image` provides an implementation of the algorithm described in Histogram of oriented gradients for human detection, whilst the `extract_fhog_features` rutines implements the Felzenszwalb's thirty-one dimensional version of HOG features, described in Object detection with discriminatively trained part based models. In this study, the thirty-one dimensional version of the HOG algorithm is used for convenience, as this routine results in a much smaller feature vector than the original implementation (Image Processing).

The Dlib library makes it possible for scientists who have very little, or no experience at all, in computer vision to experiment with feature extraction algorithms and image processing routines.

2.5 FANN C Library

Fast Artificial Neural Networks (FANN) library is a free, open-source, neural network library. It implements multilayer artificial neural networks in C, and supports for both fully connected and sparsely connected networks (FANN). Analogue to the Dlib C++ library, FANN is a helpful utility for scientists with little to no-experience in the artificial neural networks field. FANN provides a very easy-to-follow interface for Artificial Neural Network creation, training and testing. In order to experiment with a multilayer fully connected neural network, FANN provides routines that automate the process of creation, training and testing: `fann_create_standard`,

`fann_train_on_file`, `fann_save` and `fann_run` are the main steps to experiment with a neural network.

These calls allow for the creation of the network, by specifying the number of layers and the desired number of neurons within each layer; the network is created and trained using a text file with the *.data* extension, where examples of inputs and desired outputs are given to the network; afterwards, the network is saved onto a file with the *.net* extension, which can therefore be used to run the network with test data (Getting Started).

3 Image Feature Extraction

To achieve the objectives of this study, the implementation of the Felzenszwalb's thirty-one dimensional version of HOG features algorithm was used, for the task of extracting image features. The dataset for the training and testing of the network consists of a subset of the GTSRB original training dataset. Four classes of traffic signs were selected for the experiments, these being the class number 11, 14, 33 and 41. The size of the dataset is 2789 images of German traffic signs. The test dataset consists of 240 pictures of traffic signs: the neural network was tested twice; once with 240 pictures from the training dataset, and once with 240 pictures not included in the training dataset. Results on this follow later.

Thus, the process of feature extraction was reduced to iterate over the training dataset image files; processing them, with help of the `array2d` and `rgb_pixel` classes from the Dlib C++ library; all of the images where resized to 30×30 pixels width and height, in order to obtain similar dimensions in the feature vectors; and finally, call the `extract_fhog_features` routines to obtain an array of the 31-dimensional vectors. For the 30×30 pixel samples, this method call returned a 2×2 dimensional array; each element containing the 31-dimensional vector with the HOG descriptor of the corresponding block in the sampled image.

In summary, the four 31-dimensional vectors were put in a single 124-dimensional array, which correspond to the feature vector used to represent each of the images in the training dataset. For purposes of training the neural network, a routine was implemented in C to write each of the 124-dimensional feature vectors to a text file with the *.data* extension. This file follows the corresponding format showed in the example from the Getting Started page, at the FANN website.

4 Artificial Neural Network Creation and Training

The FANN C library was used to accomplish the creation and training of the neural network. After specifying the number of input neurons, intermediate layers, number of neurons on each intermediate layer, and number of outputs, the artificial neural

network is created and is ready to be trained on the training text file. This is when the text file with the *.data* extension is used.

During the training process, several parameters can be adjusted. First, the topology of the proposed neural network shall be described. It consists of 124 input neurons, corresponding each to one element of the 124-dimensional feature vector representing each image; the output layer consists of 4 neurons, which can take the value of 0 or 1: only one neuron of the output layer can have the value of 1 at a given time. Each of the output neurons corresponds to the 4 chosen German traffic signs categories: the class numbers 11, 14, 33 and 41 in the original GTSRB dataset. The number of hidden layers, number of neurons on each hidden layer, as well as the desired error, are parameters subject to experimentation.

Looking at the *.data* file gives an insight of the kind of data the network will be dealing with: the 31-dimensional HOG descriptors consist of floating point numbers between 0 and 1. The desired output for the network also consists on 0s and 1s. With this information in mind, the `FANN_SIGMOID_STEPWISE` activation function was chosen for the network, as this function deals with binary data.

Several training and testing sessions were made with different values for the hidden layers, hidden neurons and desired error parameters. Less neurons in the intermediate layer, and a lower desired error, correspond to an increased training time; whilst the more neurons and the higher desired error result in a decreased training session time. The final experiment was carried out with one intermediate layer, 32 hidden neurons within it, and a desired error of 0.00001, as these parameters showed the best results in the testing session. The results are explained in the following sections.

5 Artificial Neural Network Testing

After the training process, a text file with the *.net* file format is created, corresponding to the now trained neural network. In order to make the network perform a classification task for a given image, `fann_run` must be called, giving it the network recreated from the *.net* file, along with a corresponding image, upon which to perform the classification. This call returns an n-dimensional vector, where n stands for the number of output neurons of the neural network.

The output vector should consist of one output with the value of 1, corresponding to the selected class for the given image; and three outputs with the value of 0. Due to error in the classification process of the network, the output data contains noise: the output vector often contains floating point values close to 0 or close to 1. For the performance measure of the test process, the output neuron with the highest value (the value closest to 1) was chosen as the classification given by the network.

Two different testing sessions were carried out. In the first one, data, which has not been previously seen by the network during the training, was used. This resulted in more or less 81% accuracy in the recognition of German traffic signs, among the 240 samples given for the test. In the second test, data from the training set was

given for the task, so the network task was to classify among 240 images which it had already seen in the training phase: this test concluded with 100% accuracy in the recognition of the images. The details of the tests are given on the section below.

6 Experiment Results

After experimenting with the parameters, the following topology was chosen for the neural network: 124 input neurons, 32 hidden neurons in one intermediate layer, and 4 output neurons. Each output neuron represents one of the four chosen classes for traffic sign recognition. These parameters were found to obtain the best accuracy in the classification task. While the network is in the training process, the following data was output by the system to analyse: the training took 5107 iterations (epochs); it began with an error of 0.2510331571, and ended with an error of 0.00000998: this information is shown in Figure 4.

When testing with data which has not been previously seen by the neural network during the training, the classification performance went as follows: for the class 11, the accuracy of the classification was 81.66%; for class 14, the accuracy went to 85%; class 33 got 88.33% accuracy; and finally, class 44 went with 68.33%. This numbers add up to a total of 80.83% accuracy in the classification of German traffic signs. Figure 5 describes these observations.

It is worth noting that the training set did not have the same number of images from each class. The training set contains 1260 images that correspond to class 11; 720 images from class 14; 629 items from class 33; and only 180 pictures corresponding to class 44. This adds up to the 2789 pictures that form the training set.

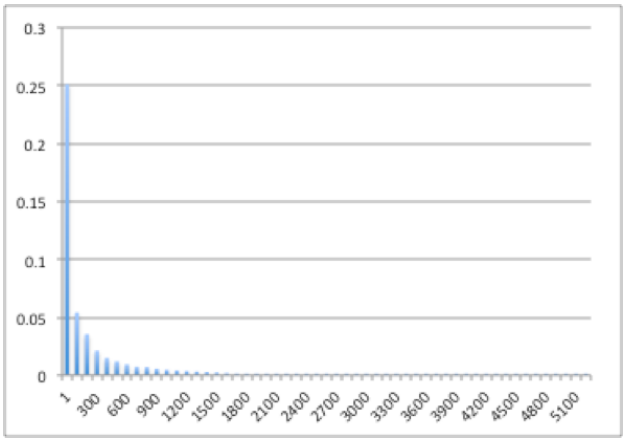


Fig. 4: Desired error versus epochs during the training session. The X-axis represents the iterations or epochs, and the Y-axis shows the desired error obtained

The relation between the number of training images and the accuracy of classification is shown in Figure 6.

The traffic sign with the classification number 44 got the lowest accuracy in the classification process; this class also corresponds to the one with fewer images on the training set. This suggests that an increase in the training data could lead to an increase of accuracy in the classification process.

For the classification task, the output neuron with the highest value was taken as the answer from the neural network. The data from the output layer had noise due to errors made by the neural network; these errors become apparent as approximations either to the value of 1 or 0. A fragment of the results data, which shows the small errors made by the neural network, is available in Table 1.

Table 1: Example of errors in the output data

File name	Class 11	Class 14	Class 33	Class 44
00000_00000.ppm	1	0	0	0.018876
00000_00001.ppm	0	0	0	0
00000_00002.ppm	1	0.278905	0	0
00000_00003.ppm	0	0	0	0
00000_00004.ppm	0.688092	1	0	0
00000_00005.ppm	0.973598	0	0	0
00000_00006.ppm	0	0	0	0
00000_00007.ppm	1	0	0	0

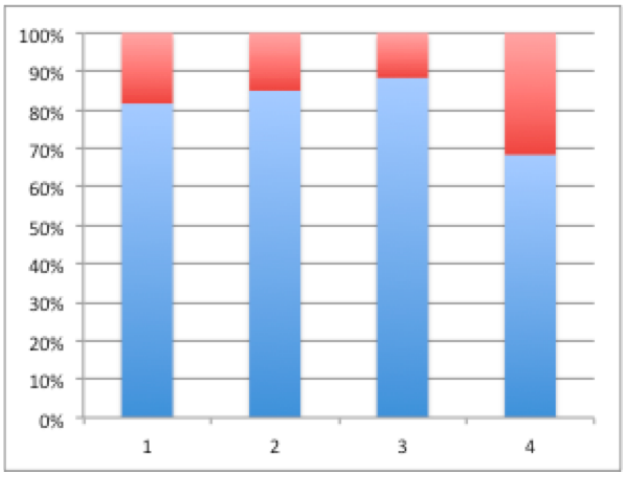


Fig. 5: Accuracy of classification versus traffic signs. The bar with the number 1 corresponds to the first class of traffic signs, class 11. The second bar corresponds to class 14, the third one corresponds to class 33, and finally is class 44

In summary, the neural network consisted of 124 input neurons, 32 intermediate hidden neurons, and 4 output neurons; each output neuron corresponding to one class of traffic sign (11, 14, 33, or 44). The dataset consisted of 2789 images taken from the GTSRB official dataset. The training set consisted of 240 images taken from the GTSRB dataset as well; 60 images of each of the four traffic sign classes. When tested with images never seen before, the neural network got 80.83% accuracy in its classification task. However, it is worth noting that, when tested with images found in the training set, the neural network had 100% accuracy in its classification task.

7 Conclusion

This study introduces an approach, based on Histogram of Oriented Gradients descriptor vectors, and artificial neural networks, to the task of classifying traffic sings. The steps taken to the accomplishment of the goal converged into satisfactory results, for experimental purposes; more than 80% accuracy in the classification task when testing with unknown data for the network, and close to 100% accuracy when testing with known data. Nevertheless, the emerging new technologies demand much more accurate results, as well as speed of execution, to properly apply a traffic sign classification system in a real-world case scenario; for example, real time classification of traffic signs.

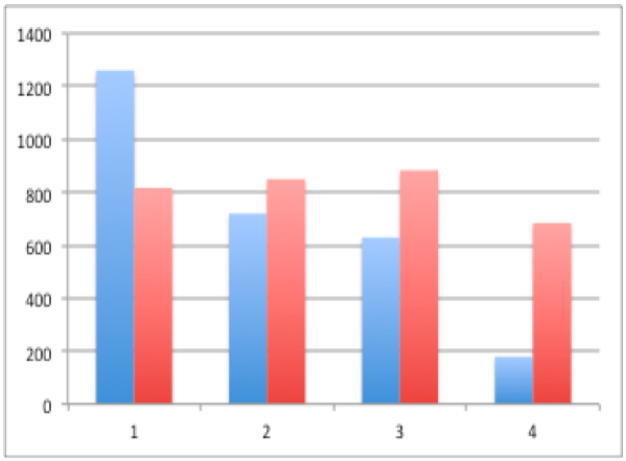


Fig. 6: Relation between number of training images and accuracy of classification. The X-axis enumerates the four chosen classes of traffic signs: class 11, 14, 33 and 44. The Y-axis corresponds to the number of images in the training set. The blue bars, show the number of images for each class of traffic sign; the red bars, show the accuracy multiplied by one-thousand

Future work that could lead to improved accuracy of classification and speed of execution is: expanding the training and testing datasets, as the more images available for training lead to an improved accuracy of the classification; to experiment with different image feature extraction algorithms; to experiment with different number of intermediate layers in the neural network; and, aid the computer program with the inclusion of new algorithms in intermediate steps of the 3-phase approach.

References

1. Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (n.d.). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. ScienceDirect.
2. Zwass, V. (n.d.). Neural network. Britannica Academic Edition.
3. Xiaoguang, H., Xinyan, Z., Deren, L., & Hui, L. (n.d.). Traffic sign recognition using scale invariant feature transform and svm. ISPRS Journal of Photogrammetry and Remote Sensing.
4. Anil, J. K., Jianchang, M., & Mohiuddin, K. M. (n.d.). Artificial neural networks: a tutorial. IEEE Xplore.
5. Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. IEEE Xplore.
6. FANN. (n.d.). Retrieved November 2013, from FANN Fast Artificial Neural Network Library: http://leenissen.dk/fann/wp/
7. Felzenszwalb, P. F. (n.d.). Object detection with discriminatively trained part-based models. IEEE Xplore.
8. Getting Started. (n.d.). Retrieved November 2013, from FANN Fast Artificial Neural Network Library: http://leenissen.dk/fann/wp/help/getting-started/
9. Image Processing. (n.d.). Retrieved November 2013, from Dlib C++ Library: http://dlib.net/imaging.html
10. McCulloch, W. S., & Pitts, W. (n.d.). A logical calculus of ideas immanent in nervous activity. Springer.
11. Major Features. (n.d.). Retrieved November 2013, from Dlib C++ Library: http://dlib.net

Traffic Signal Recognition using HoG Features in a Multilayer Perceptron

Allan Roger Reid

Abstract

Traffic signal recognition has been identified as an important test of the robustness and accuracy of a pattern recognition algorithms. The reason for this is the need to be able to benchmark improvements in the algorithms used for this sort of pattern recognition; a skill that human beings are inherently good at. The (German Traffic Sign Recognition Benchmark) GTSRB repository [3] provides a single large repository of traffic signal information thus facilitating the training and testing of new recognition algorithms on a common data set.

1 Previous approaches

From research in various papers on the subject of Traffic Sign Detection and Recognition, it was determined that the majority of the most successful attempts at this task have divided the work into the object detection, object classification and object recognition. Although the subdivision of these focuses may differ from project to project, the essence remains the same. First the traffic sign is *detected*, transformed and extracted from an image then the extracted image is *recognized* as being from a certain *classification* of traffic signs. The most popular methods were color segmentation by thresholding, color component extraction and Histograms of Gradient (HoG) or Linear Binary Pattern (LBP) construction. These methods differed in the color field used as each attempt to approximate human vision or reduce the effect of illumination [6]. These methods were supported by other methods of image manipulation such as image transformations such a binary transformations, canny image dilation, various innovative methods of shape detection and finally image normaliza-

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201, e-mail: A00354409@itesm.mx

tion[6]. Some curious approaches deliberately perturbed the images [5]. These detection methods were the followed by classification methods such as Support Vector Machine methods, k nearest neighbors (KNN) and more recently, neural networks among which were noted multilayer perceptrons (MLP), spiking and convolutional neural networks.With respect to the neural networks, noted was the need for high performance hardware to achieve the results quoted in certain papers.

2 Successes

In terms of detection, the most promising results were noted with data representation using HoG+LBP+Huehist features, HoG+Huehist features, CIELab+HoG features and CIELab-HoG+CIELab-LBP features combination in conjunction with certain classification methods, in this case Support Vector Machines. [9] Classification was best achieved with convolutional neural networks [1]. These convolutional neural networks were the winning method in the German Traffic Sign Recognition Benchmark competition of 2011 [7]. The success noted in the field do not indicate with much detail the detection methods used however. Also, the hardware used to achieve such results is not readily accessible to casual investigators. The winning team used a system with Core i7-920 (2.66GHz), 12 GB DDR3, and four graphics cards: 2 x GTX 480 and 2 x GTX 580. [1]

3 Methodology and Experimentation

This project pursues the most complete **classification** and subsequent **recognition** of 8 separate types of traffic signs stored in the GTSRB repository, using an OpenCV implementation of a Multilayer Perceptron. The traffic sign recognition will then be demonstrated as a means of bridging the gap between the academic and practical aspects of this technology.

3.1 Data Representation with HoG Features

The step of object detection is overlooked in this particular project since well identified traffic signs have already been made available in the GTSRB database. However, this raw data must be analysed for the appropriate number of key features that will then be useful to the classification algorithm. HoG (Histogram of Oriented Gradient) features were chosen based on the successes of previous successful methods of overall traffic signal recognition. The idea used by Dalal [2] is that *"local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions"*. The image window is divided into

cells and in each cell a single-dimensional histogram of edge directions over the pixels is obtained.

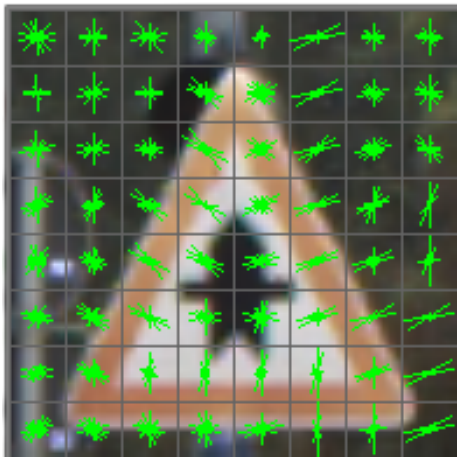


Fig. 1: HoG Features superimposed on image of Right of Way sign

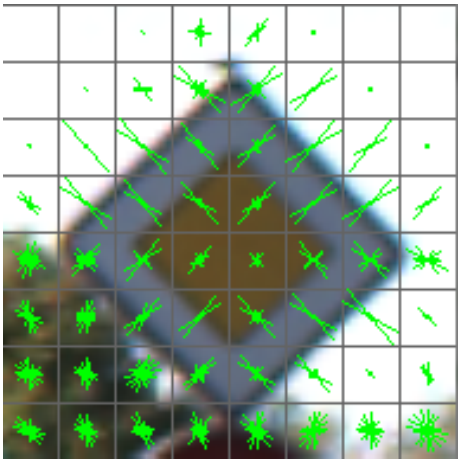


Fig. 2: HoG Features superimposed on image of Priority Road sign

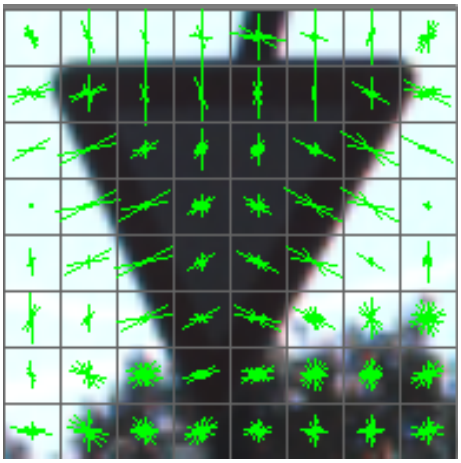


Fig. 3: HoG Features superimposed on image of Yield Right of Way sign

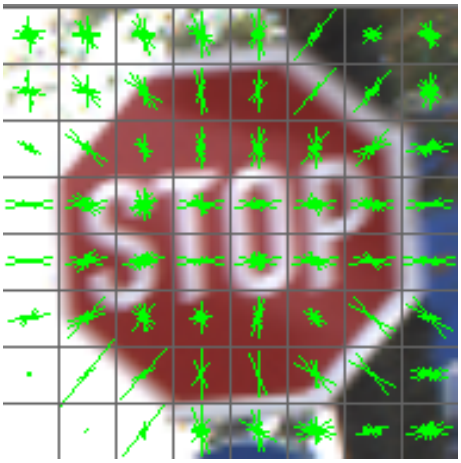


Fig. 4: HoG Features superimposed on image of Stop sign

The above images show examples of the HoG features superimposed on the images which produced them.

Per Dalal [2], the advantages of the HoG representation are as follows:
1.- it captures edge/gradient structure in a way characteristic of the local shape
2.- it is controllably invariant to geometric/photometric transformations. This is relative to the bin size.

In this project, the image is preconverted to grayscale before applying the HoG algorithm, in order to eliminate the need to perform it on each color channel. Based on Dalal's research [2], the HoG descriptor parameters, as implemented in OpenCV are set as the following:

parameter	value
winSize	Size(64,64)
blocksize	Size(16,16)
blockStride	Size(8,8)
cellSize	Size(8,8)
number of bins	9 or18
derivAper	0
winSigma	-1
histogramNormType	0
L2HysThresh	0.2
gammal correction	false
nlevels	64

Table 1: HoG descriptor parameters

With this setup, 1764 features are produced per image. The HoG descriptor and the class of traffic sign that it represents are then fed into the next classification phase as one of the rows in the entire training set.

3.2 Data Classification with Multilayer Perceptron

As a means of classification in this project, first a multilayer perceptron is trained using the HoG features previously extracted from the image set. Once training has been achieved, the degree of accuracy is measured by running the test data, with previously extracting features, through the same multilayer perceptron, this time in classification mode.

A multilayer perceptron is composed of linear classifiers based on a certain predetermined threshold. [8] This threshold is dependent on the threshold function of the neuron. In this project, a sigmoidal function is used to ensure that the hypothesis is a continuous and derivable function therefore allowing more predictable results even with noisy data. The hypothesis of the data classification, given the outputs x_j of the layer n and w_j a vector of weights assigned to each x is a vector of input values, the outputs y_i of the layer $n+1$ considering this sigmoidal function, f, can be understood as:

$$u_i = \sum_j (w_{i,j}^{n+1} \times x_j) + w_{i,bias}^{n+1} \tag{1}$$

$$y_i = f(u_i) \tag{2}$$

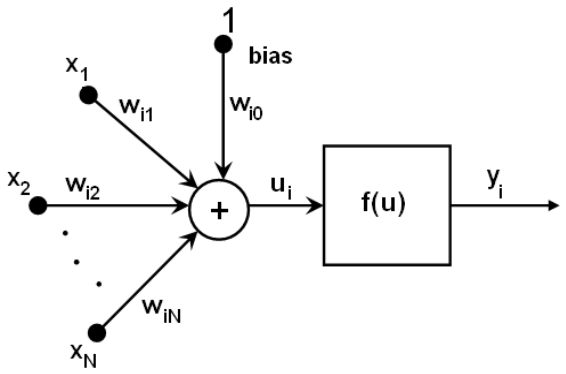


Fig. 5: Graphical representation of the neuron model

The data must be, in some way, linearly separable so that the result of this function predict the class that a set of input values belong to.

To determine the set to weights that fulfil the hypothesis, a rule similar to that used in linear regression, the perceptron learning rule, is used as applied to the sigmoid function. [4] Using a gradient descent calculation, each weight is iteratively updated with the derivative of the square difference between output and hypothesis with respect to the weight vector. This iterative updating is called back propagation. See Algorithm 1.

This rule ensures that:

- if the hypothesis is correct, the weight is not modified
- if the output is greater that the hypothesis then the weight is increased
- if the output is lesser that the hypothesis then the weight is decreased

In this project the multilayer perceptron chosen was based on the C++ class CvANN_MLP which is defined in the Machine Learning library of OpenCV 2.4.5. [8]. The neural network was implemented was run on machine with an Intel (R) Core (TM) i5 CPU @2.40 GHz per core. The program used under 25

The network was created with the function void CvANN_MLP::create(const CvMat* layerSizes, int activateFunc = CvANN_MLP::SIGMOID_SYM, double fparam1 = 0, double fparam2 = 0) with arguments:

- $n \times 1$ matrix for an index of the neural layers

The training parameters of the class were setup using the constructor CvANN_MLP_TrainParams :: CvANN_MLP_TrainParams(CvTermCriteria term_crit, int train_method, double param1, double param2 = 0) with arguments to:

- terminate only after 10000000 iterations and an error of 0.000001
- use backpropagation (see constant CvANN_MLP_TrainParams::BACKPROP)
- activation function parameter $\alpha = 0.1$
- activation function parameter $\beta = 0.1$

Algorithm 1 The back-propagation algorithm for learning in a multilayer network

function BACKPROPAGATIONLEARNING(*network*, *examples*)
  returns a neural network
  Input:
    network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
    examples, a set of input/output pairs (x, y)
    α, learning rate
  Local variables: Δ, a vector of vectors, indexed by the relevant network node
  repeat
    for each weight $w_{i,j}$ **in** *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    end
    for each example (x, y) **in** *examples* **do**
      ▷ Propagate the inputs forward to compute the outputs
      for each node i **in** input layer **do**
        $a_i \leftarrow x_i$
        for $l = 2$ to L **do**
          for each node j **in** layer l **do**
            $in_j \leftarrow \sum_i w_{i,j} a_i$
            $a_j \leftarrow g(in_j)$
          end
          ▷ Propagate deltas backwards from output layer to input
          for each node j **in** output layer **do**
            $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
          end
          for $l = l - 1$ **to** 1 **do**
            for each node i **in** layer l **do**
              $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
            end
          end for
          ▷ Update every weight in network using deltas
          for each $w_{i,j}$ **in** network **do**
            $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
          end
        end for
      end
    end
  until stop criterion is satisfied
  return network
end function

The rest of the parameters are the inputs which were modified in the different tests:

- HoG training data
- the classes of this training data
- a matrix of initializing weights
- a constant indicating that all training samples are to be analyzed

As defined earlier, the activation function used is a symmetrical sigmoid (CvANN_MLP::SIGMOID_SYM):

$$g(x) = \beta \times (1 - e^{-\alpha x})/(1 + e^{-\alpha x}) \tag{3}$$

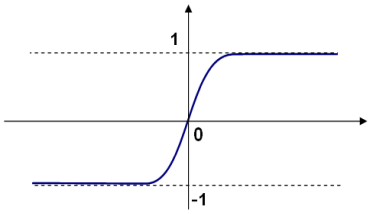


Fig. 6: Bipolar sigmoid activation function

4 Results of Image Classification and Recognition

A battery of tests was carried out using the HoG features generated, as input to the Multilayer Perceptron. Once trained, a separate set of test data was used to verify the correctness of the neural network. During preliminary tests, it was noted that decreasing the quality of the images decreased the quality of the recognition. Therefore only the largest images were used from the traffic sign database.

	Hidden neuron layers [×1764]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	2				149	14
2	4				253	13
3	6				300	16
4	8				487	23
5	10				600	25
6	10	2			**Fail**	**Fail**
7	8	2			**Fail**	**Fail**
8	6	2			2116	26
9	6	4			**Fail**	**Fail**
10	4	4			1338	29
11	2	2	2		9139	100

Table 2: Test results against 80 images for the neural network trained against 212 examples of 4 traffic signs, using 1764 HoG features

The above table shows that 100% recognition was attained with a neural network trained using 212 examples of 4 traffic signs with 1764 HoG features each. This degree of training and recognition took 9193 [s], an outlier in all the tests performed. This was also the highest level of recognition achieved in any test. Note that in three of these cases, the system did not have enough memory to construct the network, thus the 0/0 time/recognition.

	Hidden neuron layers [×3528]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	2				581	15
2	4				889	29
3	6				1332	45
4	8				1764	68
5	10				**Fail**	**Fail**
6	10	2			**Fail**	**Fail**
7	8	2			**Fail**	**Fail**
8	6	2			**Fail**	**Fail**
9	6	4			**Fail**	**Fail**
10	4	4			**Fail**	**Fail**
11	2	2	2		**Fail**	**Fail**

Table 3: Test results against 80 images for the neural network trained against 212 examples of 4 traffic signs, using 3528 HoG features

Exploration into the use of more features followed. However, the use of more features made the neural network more resource intensive to create. There are many more instances of 0/0 time/recognition due to this. In the two previous tests, the simpler compositions of neural architecture allowed for a higher percentage of recognition; a 1764:3528:3528:3528:1 [layer] neural network *(i.e. 1764 neurons in input layer, 1 neuron in output layer and 3528 neurons in layers 2-4)* and a 3528:7056:1 [layer] neural network achieved the top 100% and 30% recognition respectively.Note that rounding was used to cluster the neural network prediction to a specific class, for example, if an image was recognized to have a class of 11.3, it would be considered to be in the class of 11. These integer values were mapped to actual traffic sign classes.

	Hidden neuron layers [×1764]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	0.5				58	36
2	0.25				24	36
3	0.17				26	38
4	0.13				67	68
5	0.10				31	14
6	0.10	0.5			41	14
7	0.13	0.5			35	14
8	0.17	0.5			39	14
9	0.17	0.25			143	34
10	0.25	0.25			98	14

Table 4: Test results against 80 images for the neural network trained against 212 examples of 4 traffic signs, using 1764 HoG features, and less neurons

In order to determine if there was no problem of over-fitting, a lesser number of neurons in the network architecture, was tested. Although the time take to train the

network was also lesser, no discernible improvement in the accuracy of the resulting classification was noted.

	Hidden neuron layers [×1764]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	1	1			27	36
2	2	1			26	36
3	4	1			26	38
4	6	1			23	68
5	8	1			27	14
6	10	1			26	14
7	1	10			29	14
8	2	10			30	14
9	4	10			**Fail**	**Fail**
10	4	8			**Fail**	**Fail**
11	4	6			**Fail**	**Fail**
12	5	5			36	98
13	6	5			**Fail**	**Fail**
14	6	4			33	132
15	6	4			**Fail**	**Fail**
16	6	2			35	124
17	8	2			39	131
18	10	2			**Fail**	**Fail**
19	8	0.2			38	147
20	10	2			29	132
21	10	1			34	107

Table 5: Test results against 80 images for the neural network trained against 324 examples of 8 traffic signs, using 1764 HoG features

The aim of the project was to determine if 8 classes of traffic signs could be trained and effectively recognized by a neural network. Again, it is noted that a simpler architecture was able to achieve the highest level of recognition; a 1764:1764:3528:1 [layer] neural network, 39% recognition. As seen before, more complex network architectures were untrainable due to memory restrictions. For example on trying to train a network with 1764:7056:17640:1 [layer] architecture, the error *'Failed to allocate 1095683964 bytes'* was displayed.

	Hidden neuron layers [×3528]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	2				134	29
2	4				261	43
3	6				**Fail**	**Fail**
4	2	2			**Fail**	**Fail**
5	2	1			259	48
6	1	1	1		192	32
7	2	1	1		302	53

Table 6: Test results against 100 images for the neural network trained against 324 examples of 8 traffic signs, using 3528 HoG features

With the failure of the project to accurately classify the 8 classes of traffic signs with 1764 HoG features, and the knowledge that a lesser number of neurons did not improve this accuracy, a test with an increased number of 3528 features was performed. However, there still was no increase in accuracy of classification.

	Hidden neuron layers [×1764]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	2				46	13
2	4				77	20
3	6				101	32
4	8				130	31
5	10				157	35
6	10	2			134	25
7	2	4			163	27
8	2	6			221	49
9	2	8			280	56
10	2	10			**Fail**	**Fail**
11	2	9			**Fail**	**Fail**
12	2	8			**Fail**	**Fail**
13	2	6			320	62
14	4	6			**Fail**	**Fail**
15	4	4			311	51
16	2	2	2		155	27
17	2	2	4		200	44
18	2	2	6		219	40
19	2	2	8		**Fail**	**Fail**
20	2	4	6		**Fail**	**Fail**
21	2	4	4		**Fail**	**Fail**
22	2	4	2		263	44
23	4	2	2		241	42
24	2	2	2	2	234	38

Table 7: Test results against 84 images for the neural network trained against 233 examples of 5 traffic signs, using 1764 HoG features

Being unable to train the network successfully to recognize 8 signs, an attempt was made to train a slightly smaller number of signs. Training of 5 signs was only marginally successful with the highest recognition of 32% with architectures 1764:7056:7056:1 [layer], 1764:3528:3528:3528:1 [layer] and 1764:3528:3528:7056:1 [layer] neural network. However, on closer examination, and indeed for some earlier test cases, this classification had no diversity. This means that there was a constant numerical output pertaining to one specific class of traffic sign only, when attempting to test the classification.

	Hidden neuron layers [×3528]					
Try	Layer 1	Layer 2	Layer 3	Layer 4	Time [s]	Accuracy %
1	2				126	61
2	3				236	11
3	4				236	45
4	5				311	22
5	6				**Fail**	**Fail**
6	2	2			358	36
7	2	3			**Fail**	**Fail**
8	2	2	2		**Fail**	**Fail**
9	2	1	1		286	36

Table 8: Test results against 84 images for the neural network trained against 233 examples of 5 traffic signs, using 3528 HoG features

Recognition was improved for 5 traffic signs with the highest recognition of 61% with architectures 3528:7056:1 [layer] architecture.

5 Practical Demonstration

In the analysis of the data in the previous section, static classification of images in a predefined test set were used as a benchmark of the accuracy of the trained neural network. In the case of 4 traffic sign classes, where the recognition accuracy was 100%, the neural network weight configuration was saved and again loaded for separate dynamic tests. The size of the saved file was 920 MB, the size attributable to the large number of neurons, 12349 in the entire network

The images show the raw images captured by the camera, the HoG features superimposed on a the modified images, the class number and class image as classified by the neural network.

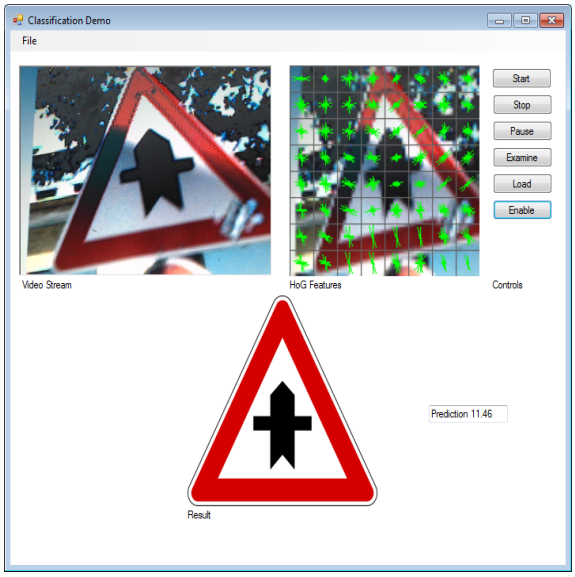


Fig. 7: Results of Dynamic Test - Right of way Sign

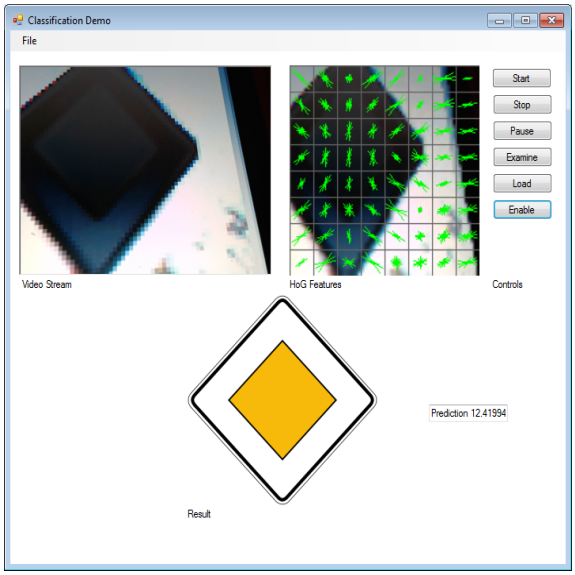


Fig. 8: Results of Dynamic Test - Priority Sign

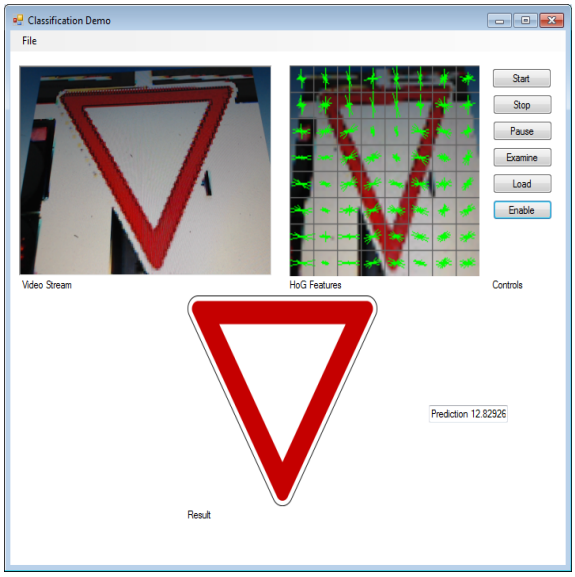


Fig. 9: Results of Dynamic Test - Yield Sign

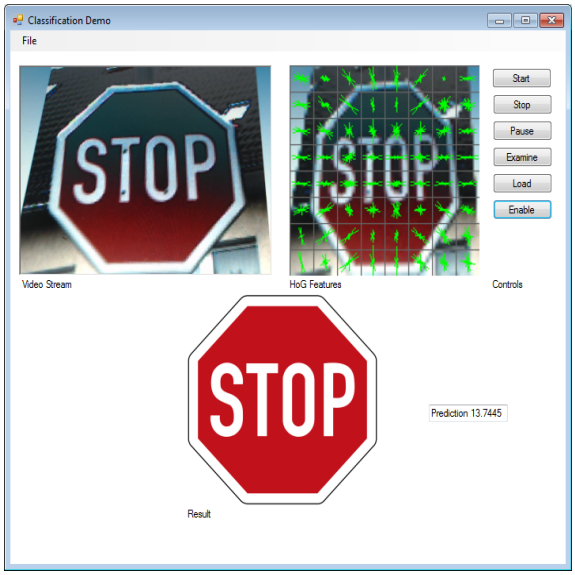


Fig. 10: Results of Dynamic Test - Stop Sign

In the above recognition of the Right of way, Priority, Yield and Stop sign, effects such as background noise, very slight occlusion and very slight rotation do not impair the neural network from successfully identifying the image. The HoG features are sufficient to just correctly classify the sign. Moving the camera to centralize the image or to make it more vertical improved the correctness of classification.

6 Conclusions

It was noted that the neural network with most success was that with architecture of input 1764 HoG features, 3 hidden layers of 3528 neurons each and an output layer of 1 neuron. 100% recognition of 84 images of 4 traffic sign classes was achieved in 9139 [s]. The recognition accuracy was proven with a dynamic test using input fed into the neural network from a web camera. In this project, difficult was encountered with configuring and training the neural network to improve accuracy in subsequent classification exercises. This included:

- the inability to change the neural architecture in a strictly systematic way as a means of discovering more accurate network configurations, since memory constraints prevented certain configurations from being created
- the number of layers used in the architecture of the neural network affected its accuracy. A network with 1 hidden layer of 17640 neurons performed 4 times worse than a network with 3 hidden layers of 3528 neurons each.
- the time taken to perform certain tests was in the order of hours. This prevented quick and repeated tests to changing the configuration of the network architecture.
- the images available in the GTSRB database are largely of low quality. Therefore since it is known that a low quality image causes the training of a noisy classifier, the majority of the low quality images were filtered from training. This reduced the total set of training images from thousands to mere hundreds.
- it is still inconclusive if a network with few training examples but examples of high quality, or many examples of low quality is the best means of training a neural network for high accuracy. With few examples of high quality the results were erratic, with the breakthrough in recognition success not predicted. A comparison of the current project methodology, with one where many low quality examples are used, is in order.

References

[1] D. Ciresan et al. “A committee of neural networks for traffic sign classification”. In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. 2011, pp. 1918–1921. DOI: 10.1109/IJCNN.2011.6033458.

[2] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[3] Editor Gildardo Sanchez-Ante. “Sistemas Inteligentes: Reportes Finales 2012”. In: *Reporte Tecnico RT-0001-2012* (2012).

[4] Stuart J. Russell and Editor Peter Norvig. “Artificial Intelligence: A Modern Approach”. In: *Third Edition* (2010).

[5] P. Sermanet and Y. LeCun. “Traffic sign recognition with multi-scale Convolutional Networks”. In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. 2011, pp. 2809–2813. DOI: 10.1109/IJCNN.2011.6033589.

[6] D. Soendoro and I. Supriana. “Traffic sign recognition with Color-based Method, shape-arc estimation and SVM”. In: *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. 2011, pp. 1–6. DOI: 10.1109/ICEEI.2011.6021584.

[7] J. Stallkamp et al. “The German Traffic Sign Recognition Benchmark: A multi-class classification competition”. In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. 2011, pp. 1453–1460. DOI: 10.1109/IJCNN.2011.6033395.

[8] OpenCV dev team. “Neural Networks”. In: *OpenCV 2.4.5.0 documentation OpenCV API Reference ml. Machine Learning* (2013).

[9] Xiaoqing Yang, Yanyun Qu, and Suwen Fang. “Color fused multiple features for traffic sign recognition”. In: *Proceedings of the 4th International Conference on Internet Multimedia Computing and Service*. ICIMCS ’12. Wuhan, China: ACM, 2012, pp. 84–87. ISBN: 978-1-4503-1600-2. DOI: 10.1145/2382336.2382360. URL: http://0-doi.acm.org.millenium.itesm.mx/10.1145/2382336.2382360.

The German Traffic Sign Benchmark with Varying Hidden Node Numbers in Artificial Neural Network

Benjamin Villegas Medina

1 Background

The German Traffic Sign Benchmark (GTRSB) is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. [**1**]

1.1 Overview

GTRSB group provide images for German signs that we used in this experiment. In total 3,515 of these images were used for training and 1,114 images used for testing the Artificial Neural Network (ANN). The GTRSB group also provide precalculated histogram of hue values (explained in section 1.1.4.1), grouped in sets for each traffic sign. [**1**]

1.2 Objective

The objective for this experiment was to evaluate the recognition rate of German traffic signs using Artificial Neural Networks (ANN), and to compare the results using ANNs with different number of neurons. We evaluated the recognition rate of four Artificial Neural Networks using 3 hidden layers, the fist AAN with 10 neurons each hidden layer, the second ANN with 50 neurons each hidden layer, the third ANN with 100 neurons each hidden layer and the last ANN with 190 neurons first hidden layer, 130 second hidden layer and 70 neurons third hidden layer.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

For this experiment we selected four traffic German signs and we used a set of images provided by the GTRSB group from the ”Institut fr Neuroinformatik”.

In Order to facilitate the evaluation we used the HUE histograms pre-calculated values from the sets of images provide by the GTRSB group; GTRSB group used the process explained in section 1.1.4.1.

The Artificial Neural Network was implemented using FANN-C library (Fast Artificial Neural Network Library).

Selected Traffic German Signs: [**7**]

1.3 Image format

- The images contain one traffic sign each. Examples in figure 1.1.
- Images contain a border of 10 % around the actual traffic sign (at least 5 pixels) to allow for edge-based approaches.
- Images are stored in PPM format.
- Image sizes vary between 15x15 to 250x250 pixels.
- Images are not necessarily squared. [**1**]

1.4 Pre-calculated Hue Histogram values

Histogram of hue values for each single image. Images were not scaled, but a fixed border of 5 pixels was ignored for computation of histograms. Note that we used 0 <=hue<= 255. Therefore, the resulting feature vector conventionality is 256. [**1**]

Fig. 1: Traffic images

1.4.1 What is Histogram of Hue Values?

A histogram is the representation of the gray level or color distribution in an image. One color space of the image is subdivided in a number of bins, the histogram is created by counting the number of pixels in each bin. This process results in a very large number of bins, and adjacent bins would reveal only trivial differences. Therefore and small change in the illumination or shadows or the presence of noise generate that a large number or pixels generate high variances between the bins. Then two similar images may have very different histograms representations. [**4**] [**5**]

The hue values are weighted calculating the standard deviation of the different areas of the image using the RGB tristimulus. The hue component contains most of the color information; therefore it is almost constant regardless of the changes in the illumination conditions. [**4**] [**5**]

Assuming that the standard deviation (s*) is normalized between 0 and 1, the chromatic image region can be defined as a fuzzy set with the membership function given by the following S type function: [**4**]

$$\mu(s^*) = \begin{cases} 0 & \text{if} \quad 0 \leq s^* < a \\ 2(\frac{s^*-a}{b-a})^2 & \text{if} \quad a \leq s^* < \frac{a+b}{2} \\ 1-2(\frac{s^*-b}{b-a})^2 & \text{if} \quad \frac{a+b}{2} \leq s^* < b \\ 1 & \text{if} \quad b \leq s^* < 1 \end{cases} \tag{1}$$

Where a and b are two given sub-unitary values with $a < b$.[**4**]

The weight values are determined based on s^* value in that pixel using the membership function (1.1) as follows:[**4**]

$$w_h(n,m) = \mu(s^*(n,m)) \tag{2}$$

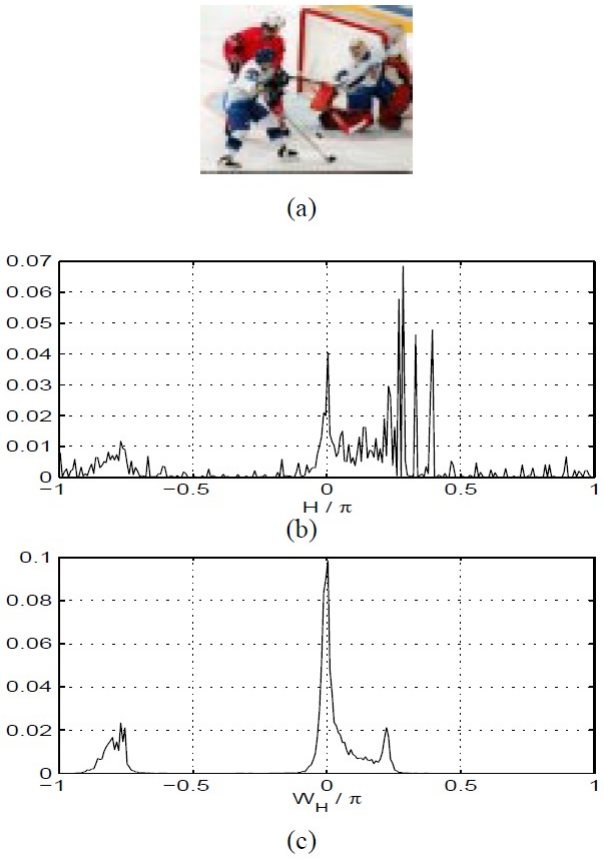
The weighted histograms are then created using the weight values defined in (1.2). Where L_H denote the number of bins allocated for hue, where.[**4**]

$$H(n,m) \in \{0,1,...,L_H - 1\} \tag{3}$$

The weighted hue histogram W is defined as:

$$W(l) = \frac{\sum_{n,m} w(n,m)\delta(H(n,m)-1)}{\sum_{n,m} w(n,m)} \tag{4}$$

For each $l = 0,1,...,L_h 1$, where delta is the Kronecker delta function.[**4**]



(a)

(b)

(c)

Fig. 2: The figure shows: the color image (a), the hue histogram (b), the weighted hue histogram (c). [**4**]

1.5 Artificial Neural Network

Artificial neural network (ANN) is a model that simulate the human brain neural network. An ANN like a human brain learn with examples. This mean that an ANN have a function to learn from examples that are different in position, color intensity, light, etc; but is the same object. One example is recognize the face of a person in different pictures: one picture in the beach, another one in a city during the night, etc. [**3**] [**6**]

When we want to the ANN learn a problem then the function to learn need a set of inputs and outputs and the way that this function should work. Example is the XOR function that have two input parameters and one output. This is a simple problem and with an image recognition the input will be the Image features (previously explained) and the output could be a value that represents the approximation to the target object; this value can be a percentage (from 0 to 1) or a boolean value. [**3**]

Artificial neurons are similar to their biological human neurons. These have connections between them with input connections witch are used to determine the strength of their output. The output is an Activation function that use the inputs to generate a weight value with a number between 0 (for low input values) and 1 (for high input values). The output value is send as input for other neurons. [**6**]

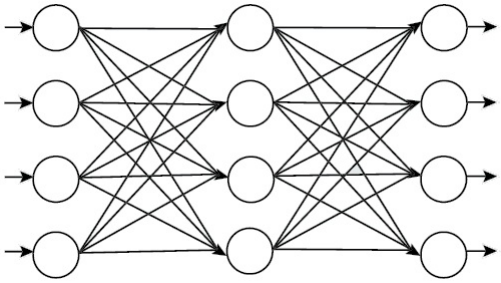


Fig. 3: ANN with four input neurons, a hidden layer and four output neurons

ANN has the neurons ordered in layers with connections between the layers. The first layer contains the input neurons and the last layer contains the output neurons. These input and output neurons represent the input and output variables of the function. Between these layers we have hidden layers with connections. We train the ANN to learn the neurons store information with the weights and the functions is adjusting with the hope that it will give a correct output. [**6**]

2 Problem Definition

A person who drives is capable to recognize all the variety of traffic signs around the road with close to 100% effectiveness. This is not only for traffic signs, also apply for other kind of images. [**1**]

For an intelligent agent the recognition of traffic signs is a challenging real-world problem of high industrial relevance. Although Several Studies for this topic has been published and there are commercial systems in the market; it is missing a systematic comparisons of different approaches. [**1**]

Applications of an intelligent agent that recognize traffic signs can be installed in a car to alert the diver when is over-passing the speed limit, is entering to a school zone, or if there is a danger sign then the driver needs reduce the speed, etc. In this way reduce the risk for accidents.

Complexity in the traffic sign recognition is that traffic signs can provide a wide range of variations between classes in terms of , size, color, shape and text. Another complexity is the large variations in visual appearances due to illumination changes, partial occlusions, rotations, weather conditions, etc. [**1**]

3 Previous Work

GTSRB group in the The International Joint Conference on Neural Networks 2011 (IJCNN 2011) held a competitions in two phases. Participants used the data sets of images and features provided by the GTSRB group.

Results:

- IDSIA (Istituto Dalle Molle di Studi sull'Intelligenza Artificiale) Achieves a 99.46% recognition rate. implementation of a Convolutional Neural Network (ConvNet) on Graphics Processing Units (GPUs). [**11**]

- Pierre Sermanet and Yann LeCun, The Courant Institute of Mathematical Sciences, New York University. Achieves a 98.84% recognition rate. implementation of a Convolutional Neural Network (ConvNet). The traditional ConvNet architecture was modified by feeding 1st stage features in addition to 2nd stage features to the classier using 32x32 color input images. [**12**]

- Pierre Sermanet, Koray Kavukcuoglu and Yann LeCun, The Courant Institute of Mathematical Sciences - New York University. Achieves a 98.31% recognition rate. Implementation of a Multi-Scale Convolutional Neural Network using unsu-

pervised pre-training with Convolutional Predictive Sparse Coding (ConvPSD). The ConvNet was implemented using the EBLearn C++ open-source package. [**13**]

- Fatin Zaklouta and Bogdan Stanciulescu and Omar Hamdoun. Achieves a 96.14% recognition rate. Implementation K-d trees and Random Forests using a Histogram of Oriented Gradients (HOG) based Support Vector Machines (SVM). [**14**]

- Rajesh, R. Network Syst. & Technol., Thiruvananthapuram, India. Achieves a 90.99% recognition rate. Use of Coherence Vector of Oriented Gradients (CVOG) Implementation with ANN. [**15**]

4 Experiment

4.1 Implementation

We used FANN-2.2.0 library to train and test the ANNs.

FANN library has a function to train the ANN from a file. The template for this file is:

- First line contains the parameters:
  - Number of sets.
  - Number of inputs.
  - Number of outputs.
- Second line contains the vector with the Hue histograms values.
- Third line the expected output.

GTRSB group provides datasets with Histograms HUE features from the images. There is one file for each image. In each file there is a Hue Histogram value per line. In order to train and test the ANN, using FANN, We created a program in $C++$ to convert all the files into a single file containing all the values in the format required by FANN library. Using this program, we created two files with the datasets; the first one contains the training dataset and the second one the testing dataset.

In order to improve the performance the output expected values where normalized. For ”stop sign” the number 1 was normalized to 0.25, for ”tractors and trucks” sign the number 2 was normalized to 0.5, for ”entry prohibited” sign the number 3 was normalized to 0.75 and ”danger” sign the number 4 was normalized to 1.

We created another two programs in $C++$ to train and test the ANNs. The first

program has the configuration parameters for the ANN and in order to train the ANN we used the FANN function fann_train_on_file, this function reads the file and train that ANN. The second program reads the training dataset file and populates the input structure to call the fann_run function who evaluates the inputs and generate an output using the trained ANN.

4.2 Configuration

The testing was made with 4 ANNs configured as follow:

	ANN 1	ANN 2	ANN 3	ANN 4
Layers	5			
Input Layer	256			
Output Layer	1			
Hidden Layers	10-10-10	50-50-50	100-100-100	190-130-70
Activation Hidden Layer	FANN_SIGMOID_SYMMETRIC			
Activation Output Layer	FANN_SIGMOID_SYMMETRIC			
Desired Error	0.0001			
Max epochs Training	500000			

4.3 Training

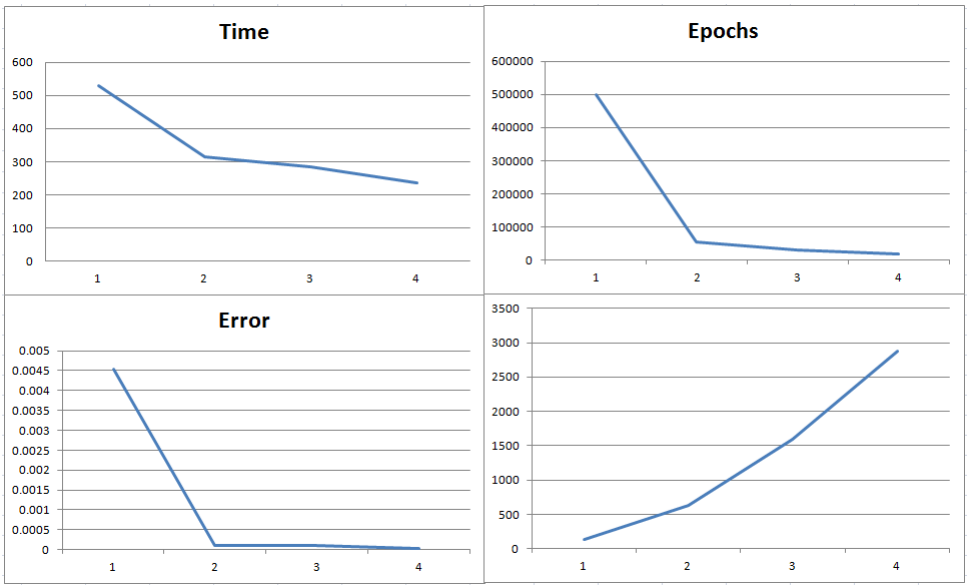
We trained the ANNs with a total of 3,514 images divided on:

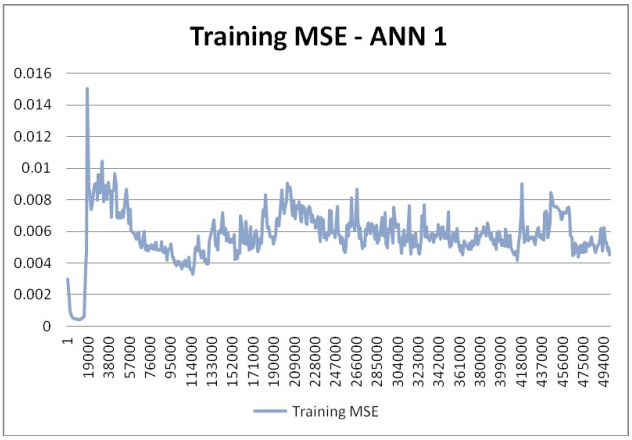
- ”Stop sign” - 781 Images
- ”Tractors and trucks” sign - 411 images
- ”Entry prohibited” sign - 1,111 images
- ”Danger” sign - 1,201 images

Results from training:

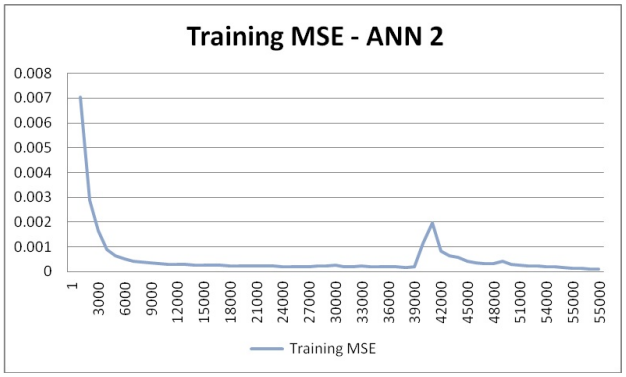
	ANN 1	ANN 2	ANN 3	ANN 4
Time	528 min	314 min	284 min	236 min
Epochs	500000	55530	30000	19000
Error	0.004539	0.000100	0.000098	0.000017
File Size	128 kb	632 kb	1,598 kb	2,884 kb

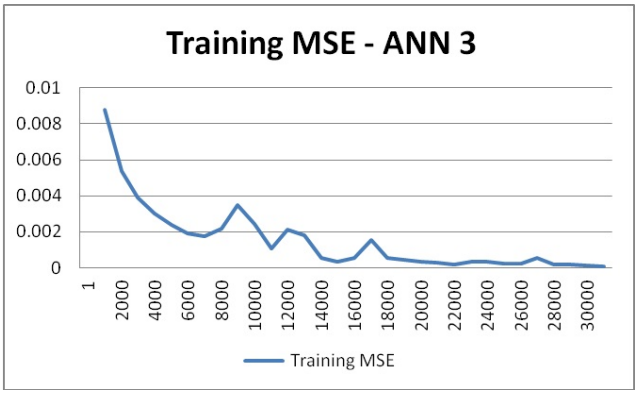
Where MSE = mean square error

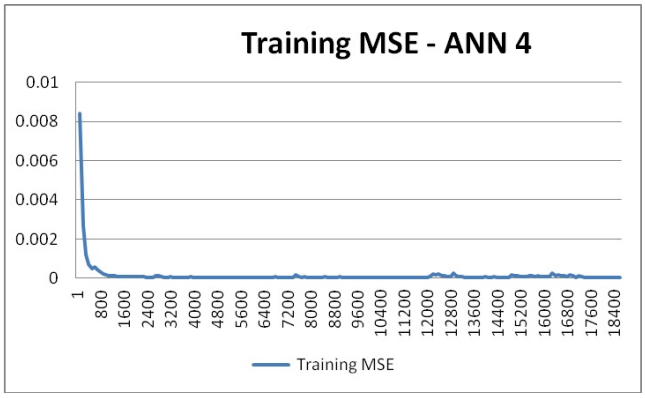




Where MSE = mean square error







4.4 Testing

We tested the ANNs with a total of 1,114 images divided on:

- ”Stop” sign - 241 Images
- ”Tractors and trucks” sign - 121 images

- ”Entry prohibited” sign - 361 images
- ”Danger” sign - 391 images

4.5 Results

To validate the effectiveness rate of the AANs, We used an error margin of 5%.

Results from Testing:

ANN 1 → 10-10-10

Image	Images	Indentified	Rate
1	240	193	80.42%
2	120	94	78.33%
3	360	287	79.72%
4	390	335	85.90%
Total	1110	909	81.89%

ANN 2 → 50-50-50

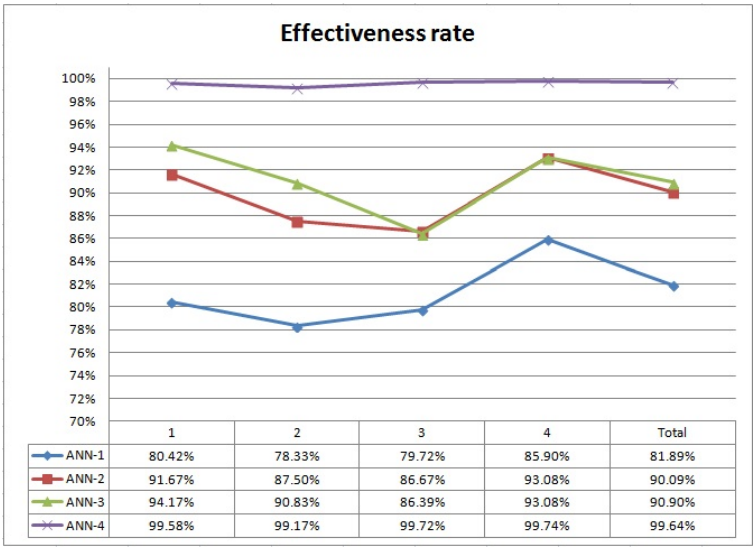
Image	Images	Indentified	Rate
1	240	220	91.67%
2	120	105	87.50%
3	360	312	86.67%
4	390	363	93.08%
Total	1110	1000	90.09%

ANN 3 → 100-100-100

Image	Images	Indentified	Rate
1	240	226	94.17%
2	120	109	90.83%
3	360	311	86.39%
4	390	363	93.08%
Total	1110	1009	90.90%

ANN 4 → 190-130-70

Image	Images	Indentified	Rate
1	240	239	99.58%
2	120	119	99.17%
3	360	359	99.72%
4	390	389	99.74%
Total	1110	1106	99.64%



	1	2	3	4	Total
ANN-1	80.42%	78.33%	79.72%	85.90%	81.89%
ANN-2	91.67%	87.50%	86.67%	93.08%	90.09%
ANN-3	94.17%	90.83%	86.39%	93.08%	90.90%
ANN-4	99.58%	99.17%	99.72%	99.74%	99.64%

5 Analysis of the results and Conclusions

We started the experiment using Haar-like features, these files contains 11,584 features per image and when we was training the ANN, this was taking 36 hours and the results in the testing were not good. Then we normalized the values using the standard deviation method and the result was better but still the training was taking so much time. Finally we decided to use the hue histograms values.

We did several testing using different number of neurons. At the beginning when we used just one hidden layer the effectiveness of the ANNs was not good, we found that using 3 hidden layers was more effective that 1 or 2 hidden layers. Also we found that using expected output values normalized between 0 and 1 was more effective. At the beginning we was training with expected output values from 1 to 4 and the ANN was generating high variances of MSE between epochs.

Regarding to the training we observed that increasing the number of neurons the time used for training decreased. This was because the number of epochs was minor but the time to process each epoch was more.

When we was training the ANN-1, we observed that the MSE had a lot of variances between epochs. When we trained the ANN-2 and ANN-3 the variances between epochs were less, and there was only some peak. And finally with ANN-4 the variances were really small, it was almost constant. We can conclude that using more neurons the ANN learn more quickly because it is easier for the learning algorithm to adjust the weights.

Regarding to the testing, as we can observe, the use of more neurons in the AAN increased the effectiveness rate. It was interesting how this was more effective with ANN-4, this one learned more quickly and the effectiveness rate was 99.6%. Comparing with the effectiveness of ANN-1 that was 81.89%, there was a great improvement.

We observed that the file size for the ANNs increased with the number of neurons and therefore the process consume more memory. The good news are that with the actual computers and with the problem we did not get any memory problems.

The results using ANN-4 were superior to the results reported in the GTRSB web site. Basically this was because we are training the ANN using only 4 traffic images. The reported results are for 42 different traffic images. we are sure that performance can be improved, increasing the number of neurons, playing with the activation functions and so on.

It was a very interesting exercise and we learned a lot. We want to continue researching with other activation functions and other ANN libraries to compare the results. Also we want to test ANN-4 with more traffic images.

References

1. Website: tallkamp, Schlipsing, Salmen & Christian, The German Traffic Sign Benchmark, 2010, url = ”http://benchmark.ini.rub.de/?section=gtsrb&subsection=about
2. Article: J. Stallkamp and M. Schlipsing and J. Salmen and C. Igel, Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, journal = Neural Networks, 2012, url = http://www.sciencedirect.com/science/article/pii/S0893608012000457
3. Book: Stuart Russell, Peter Norvig, Artificial Intelligence a Modern Approach, Third Edition, 2010
4. Article: Fillipe Souza, Eduardo Valle, Guillermo Chavez and Arnaldo Araujo, Hue Histograms to Spatiotemporal Local Features For Action Recognition, 2005
5. Article: Marius Tico, Taneli Haverinen and Pauli Kuosmanen, A Method Of Color Histogram Creation For Image Retrieval. 2010
6. Article: Steffen Nissen, Neural Networks Made Simple, 2005, url = http://fann.sourceforge.net/fann_en.pdf
7. website: USAREUR learning,Chart German Signs, url = http://www.usareurpracticetest.com/germany/documents/sign_chart.pdf
8. website: OpenCV, OpenCV 2.4.6.0 documentation, July-2013, url = http://docs.opencv.org/
9. Article: Joost van de Weijer and Cordelia Schmid, Coloring Local Feature Extraction, 2010

10. Dataset: Stallkamp-IJCNN-2011, Johannes Stallkamp and Marc Schlipsing and Jan Salmen and Christian Igel, IEEE International Joint Conference on Neural Networks, The German Traffic Sign Recognition Benchmark: A multi-class classification competition, 2011, 1453–1460
11. Article: Dan Ciresan, Ueli Meier, Jonathan Masci and Jurgen Schmidhuber, A Committee of Neural Networks for Trafc Sign Classication, 2011, url = http://www.idsia.ch/ juergen/ijcnn2011.pdf
12. Article: Pierre Sermanet and Yann LeCun, Trafc Sign Recognition with Multi-Scale Convolutional Networks, 2011, url = http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf
13. Article: Pierre Sermanet, Koray Kavukcuoglu and Yann LeCun, Trafc Signs and Pedestrians Vision with Multi-Scale Convolutional Networks, 2011, url = http://snowbird.djvuzone.org/2011/abstracts/168.pdf
14. Article: Fatin Zaklouta and Bogdan Stanciulescu and Omar Hamdoun, Trafc Sign Classication using K-d trees and Random Forests, 2011, url = http://caor.mines-paristech.fr/_media/users/ijcnn.pdf
15. Article: Ragesh N K, Neural Networks (IJCNN), The 2011 International Joint Conference on, 2011, 907–910, url = http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp& arnumber=6033318

Image Classification using Machine Learning Algorithms for the German Traffic Sign Recognition Benchmark Problem

Miguel Sanchez and Diego Torres

Abstract The following document will analyze based on empirical results the performance of different machine learning algorithms when used to solve the benchmark problem of German Traffic Sign. Motivated by the wide-range of potential applications of image recognition and in particular traffic signs, our intention is to analyze some approaches and provide strong conclusions for people who are working with the problem. For the experiments presented in this document, a Feed-Forward Neural Network trained with a Resilient Propagation Algorithm is used.

1 Purpose

Due to the recent interest of the scientific community for an automated, more efficient and intelligent transportation form, problems such as the German Traffic Sign Recognition have emerged and have taken a particular importance nowadays. Our intention with this document is to contribute to the research in this field by providing an efficient solution using two of the most popular approaches and analyzing the obtained results.

2 Introduction

First, some concepts will be exposed to give a brief introduction to the reader regarding the main topics of this project.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201. e-mail: a01093408@itesm.mx, e-mail: a01201306@itesm.mx,

*2.1 German Traffic Sign Problem[*GTSRB-online*]*

As a part of the International Joint Conference on Neural Networks on its 2011 edition, this multi-class, single image classification challenge was presented. On its web page more than 50,000 images taken from real life pictures are hosted. This provides enough material to properly train a machine learning algorithm and present an alternate solution different from the commercial available ones. The pictures presented were taken with different illumination conditions, rotations, weather conditions, etc.

The competition consists on presenting an algorithm capable of recognizing a single image between 40 different classes of traffic signs, discarding time performance and focusing on results accuracy.

*2.2 Neural Networks[*nn-article*]*

As its name suggests, these type of computation models were inspired by how a human brain works: using a network of interconnected neurons. Historically, neural networks have had three main development stages. In the 1940 decade McCulloch and Pitts proposed them for the first time. Following that, Minsky and Papert demonstrated that the model previously proposed had several limitations, discouraging several researchers to continue working on this field. And finally, the third stage happened on 1980 when Werbos proposed a new learning algorithm for neural network that led the resurgence and wide appliance inside the scientific community of this computational model. The main benefits of using a Neural Network for solving a computational problem are the following:

- Parallel processing
- Distributed computation
- Learning ability
- Generalization
- Adaptability
- Context information processing
- Error tolerance
- Low energy consumption

Due to the mentioned features, neural networks are very good at solving problems where traditional programming cannot give an efficient solution or even worse cannot provide any solution. Some of these problems are: pattern recognition, object classification, function approximation, behavioral prediction, optimization and control theory.

The basic architecture of a Neural Network has three components: input layer, hidden layers and output layer. The main job of a neuron is to calculate the weighted

sum of the input values, evaluate them with the activation function and give as a result the output value which will be used in the following neuron layer. The mathematic representation of a neuron is the following:

$$y = \theta\left(\sum_{i=1}^{n} w_i x_i - u\right) \tag{1}$$

There are two main architecture types of neural networks: feed-forward networks and recurrent networks. Feed-forward networks do not analyze previous information and responds only current data input, they are also called static. Recursive networks have loops inside of its topology making them dynamic because the output depends on current input data and previous feedback data.

Neural Networks are very useful for its learning ability. This process is called training and it is about assigning the correct weight to each neuron. There are three methods for learning: supervised, non supervised and hybrid.

- In the supervised method the neural network is provided with the correct result as part of its training data.
- On the other hand, the non supervised method does not need the correct result in order to learn. Based on the input data and the possible correlations between them adjust the neural network.
- Lastly, the hybrid method combines both supervised and non supervised methods to establish the weight values.

*2.3 Support Vector Machines[*svm-article*]*

A support vector machine is another type of machine learning algorithm proposed by Cortes and Vapnik in 1995, used specially for two-group classification problems. It is based on the idea that input vectors are non-linearly mapped to a high-dimension feature space. In other words, this hyper plane or set of hyper planes created by the algorithm will be the borderline that divides the input data into the two desired classification groups, after that the model will assign the new input data into one category or the other. The functional margin is factor that will determine the ability to classify of the Support Vector Machine and is defined as: *The largest distance to the nearest training data point of any class.* The following is the mathematical relation between a point in the feature space and the hyperplane:

$$\sum_{i} \alpha_i K(x_i, x) \tag{2}$$

There are two main types of Support Vector Machine: linear and non-linear. If the training data are linearly separable a hyperplane can be constructed to separate data in some fashion where data can be separated into two groups without any points

between them. The bigger this functional margin is, a better accuracy the SVM will have when classifying data.

Since this type of machine learning algorithm is just applicable for two-group classification problem, an extension called Multiclass SVM was proposed where this multiclass problem was reduced into multiple binary classification problems. Handwriting recognition, text categorization, image classification, protein classification are some of the applications for Support Vector Machines nowadays.

*2.4 PPM Format[*PPM-online*]*

The Portable Pixel Map Format is one of the simplest color image file formats. It has two main sections: a header and the body. The header will contain the following:

- A magic number to identify the image file type, in the case of PPM is P6.
- The image width, formatted as ASCII characters in decimal.
- The image height also formatted as ASCII characters in decimal.
- The maximum color value between 0 and 65536.

A newline character 0x0A will delimit the header from the body containing the image data. The data will be presented on a RGB format, assigning 1 or 2 bytes for each channel.

*2.5 ENCOG Framework[*encog-online*]*

Encog is a Machine Learning Framework available for Java, .Net and C/C++. It provides easy access to several algorithms such as Support Vector Machines, Artificial Neural Networks, Genetic Programming, Bayesian Networks and Hidden Markov Models. Most of Encog training algorithms are multi-threaded and make use of multi-core hardware.

3 Implementation

3.1 Feature Extraction Algorithms

3.1.1 Zernike Moments

Moments were one of our first options for feature extraction on the images.

> Moments describe numeric quantities at some distance from a reference point or axis.

There are several kinds of moments for image feature extraction. One of the most commonly used kinds of moments for image feature extraction is the Zernike Moments. Zernike Moments happen to represent an image as an image function projected onto a set of orthogonal functions. The main advantages of using this kind of moments are listed below:

- Image representation is rotation invariant.
- Accurate representation of detailed shapes.
- Information redundancy is maintained at a low level.
- Can be adapted to be scale and translation invariant.

3.1.2 Scale-Invariant Feature Transform (SIFT)

SIFT is a computer vision algorithm for describing local image features. This algorithm is mainly used in image processing for stitching, which is done by identifying the features on each one of the pieces to be merged and then matching them against each one of the contiguous pieces. This algorithm is also scale and rotation invariant.

3.1.3 Multi-Scale Oriented Patches (MOPS)

MOPS is an algorithm designed to find matches between different images based on several regions called patches positioned in several interest points which are intended to be in the *corners* or edges found in the image. This algorithm is rotation invariant.

3.2 Pre-processing

Training data samples from the German Traffic Sign Recognition Benchmark (GTSRB) project were structured as follows: The top level folder, called Images, contained 43 category folders, each one containing several PPM image files and a single CSV annotations file describing two points (x, y) representing the region of interest (ROI) in each of the PPM files in the folder. It's worth mentioning that the image

dimensions varied from 15x15 to 256x256 pixels and that the ROI was not necessarily a perfect square.

That said, we needed a way for normalizing the given samples so that a Machine Learning algorithm wouldn't have a varying sized input, as that is pretty difficult to manage. We decided to try several things and started checking several feature extraction algorithms described in the above section.

For the purposes of this paper we decided to use Java as the base implementation language, as there is a vast amount of Machine Learning libraries to test with. Unfortunately, Java library implementations of the before mentioned feature extraction algorithms were either inexistent or undocumented.

Afterwards, we went through several pieces of software which implemented the above feature extraction algorithms trying to get training data from the GTSRB images with the purpose of passing it to a Machine Learning algorithm for image recognition with no success.

Given the lack of feature extraction libraries, we decided to implement a Java application for generating a Machine Learning training file. We started researching for Neural Network image recognition implementations.

We found a project called Neuroph, a Neural Network IDE which could be trained for performing image recognition tasks. In order to perform image recognition, Neuroph needed to get a set of bitmaps containing the extracted features in JPEG format, classified in several categories. Using the Neuroph image recognition feature implied that we need to extract the features of every image in our dataset, and then transform the dataset file structure so that Neuroph could perform the task, which was time consuming given the circumstances. Perhaps, one of the tutorials in the Neuroph documentation suggested that image recognition could be done by Neural Networks by providing them with the whole image formatted as an array of integers representing the intensity values of three different channels: Red, Green and Blue (RGB). According with this tutorial, the input array had to be formatted as follows: The first third of the input should contain all red pixels in the image, the second third the green ones and the last third the blue ones. And so we did.

3.3 Normalizing the dataset

First of all, we needed a way to read all the PPM files we got from the GTSRB training dataset, so we began to search for a Java library which turned out to be unusable because it parsed the file as an array of chars, which were encoded in a different charset corrupting the image file data. So we decided to implement a PPM parser as part of our Java application. This new parser allowed us to read each one of the

image files in our dataset as a byte array avoiding encoding issues and provided us with relevant data like the image dimensions embedded within the file.

Given the CSV annotations files found in each of the category folders of the GTSRB training dataset, we should crop the image files to provide the Machine Learning algorithms with the minimum possible data so that the output accuracy could be increased. We used the OpenCSV library for parsing all CSV files in the dataset as it was widely used and useful for our purposes.

The size and ROI ratio varying image files and annotations in the training dataset was an important concern because, as said before, varying size input on Machine Learning algorithms was difficult to manage. That said, we determined to resample all images to the smallest size on the dataset, being this value 15x15 pixels. For this purpose, we used a library called Java Image Scaling, which provided us with better quality results than the Java embedded image scaling solution. This library happened to work only with the Java BufferedImage object, which was intended to represent an image, so we implemented a function for transforming our PPM file content into an IntRGB flattened array as it was one of the multiple formats accepted in the BufferedImage constructor. Once we had our BufferedImage object generated from the PPM file we scaled it to 15x15 and got the result into a new PPM which then was converted into a CSV record.

In order to generate the CSV training file for the Machine Learning algorithm, we recalled our findings on the Neuroph image recognition tutorial and implemented an export function in our PPM parser for providing all the data to the algorithm. The generated CSV records consisted of 675 columns (3 channels 15x15 pixels image) of image data plus 1 more column holding the category data for performing the image classification.

4 Experiments and Results

4.1 First Experiment

For the first experiment, we performed some training tests on files containing 6 categories. For this purpose we used the Encog Workbench tool, letting it train two Machine Learning algorithms: Neural Networks and SVM. After 11 hours and 4 minutes of training a Neural Network using the Resilent Propagation method, we realized the error was still over 79%, which was not acceptable at all. When we tested our training file against an SVM, we noticed that after one hour of training, the error got stuck on 89%, even when more training iterations were being done. So we stopped the training and decided to take another approach by simplifying the data supplied to the algorithms.

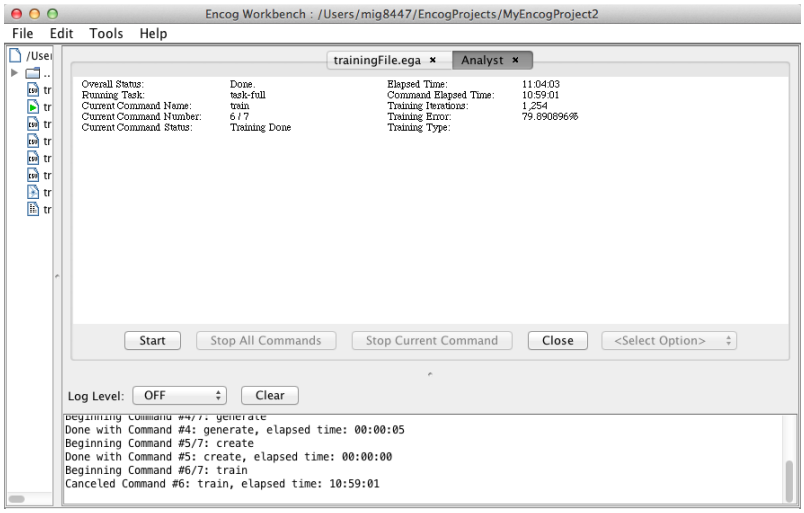


Fig. 1: Feed-forward Neural Network trained through Resilient Propagation algorithm with data containing 1 column for each color channel and 6 categories of Traffic Signs from GTSRB using ENCOG Workbench.

In order to simplify the data, we took the IntRGB array we previously had for the PPM to BufferImage conversion and wrote it as the CSV image data. This reduced the data size from 675 to 225 columns (15x15 pixels) with no data loss. The IntRGB array contained a set of integers, each of them representing the pixels in the image. In order to get each one of the integers in the IntRGB array, we performed 8-bit left shifts, one per color channel, appending each of the channel values into the integers.

In this way, the image could be reconstructed only by performing 8-bit right shifts preventing data loss.

4.2 Second Experiment

For the second experiment, we performed the training tests with the Neural Networks and SVM algorithms on the simplified training datasets with the same 6 classes as the ones used on our first experiment. On one hand, after 6 hours and 6 minutes, the error got stuck at 14.37%, less than with our first dataset. On the other hand, SVM was still getting stuck at over 80%, so we decided to discard the idea of testing with the SVM algorithm again.

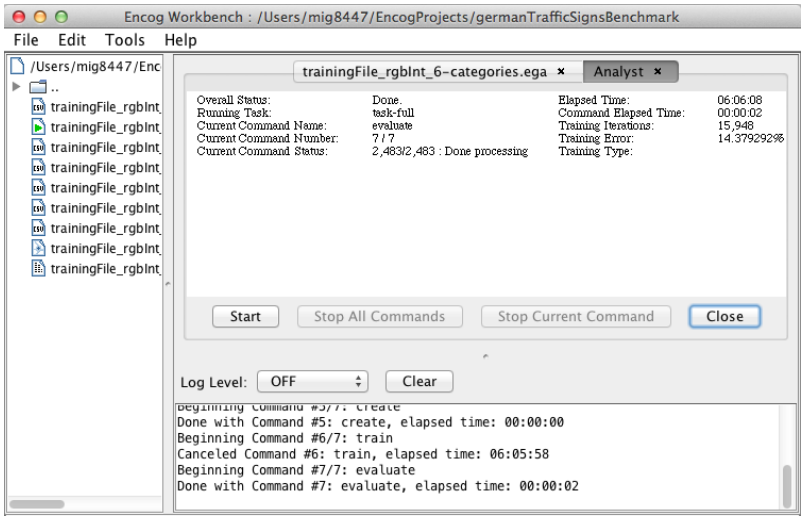


Fig. 2: Feed-forward Neural Network trained through Resilient Propagation algorithm with data containing 1 column with all the color channel combined (IntRGB) and 6 categories of Traffic Signs from GTSRB using ENCOG Workbench.

4.3 Third Experiment

The third experiment involved reducing the number of categories in the training file to 4 and testing one more time with the Neural Networks algorithm. It took 3 hours with 13 minutes for the Neural Network to get to the 9.59% of error using the using

the Resilent Propagation method once again, which we considered acceptable for this paper purposes.

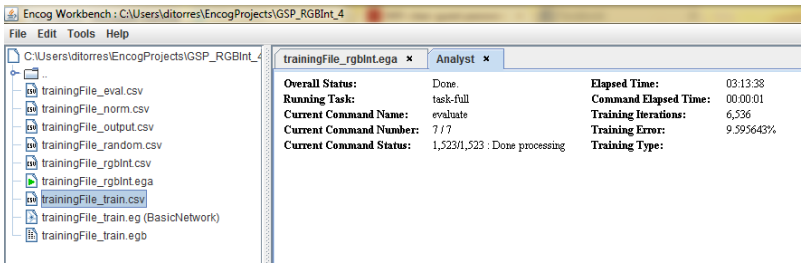


Fig. 3: Feed-forward Neural Network trained through Resilient Propagation algorithm with data containing 1 column with all the color channel combined (IntRGB) and 4 categories of Traffic Signs from GTSRB using ENCOG Workbench.

4.4 Fourth Experiment

We decided to make one last experiment and re-train the Neural Network using a different training method: Simulated Annealing. After 14 hours and 13 minutes of re-training the Neural Network, we could achieve an error of 7.68%, much better than in our previous experiment, although it took more time.

BasicNetwork

Input Count	225
Output Count	3
Encoded Length	77,176
Resettable	true
Context	true

Layers

Layer #	Total Count	Neuron Count	Activation Function	Bias	Context Target Size	Context Target Offset	Context Count
1 (Output)	3	3	ActivationTANH	0.0000	0	0	0
2	338	337	ActivationTANH	1.0000	0	0	0
3 (Input)	226	225	ActivationLinear	1.0000	0	0	0

Fig. 4: Configuration of the final Feed-Forward Neural Network.

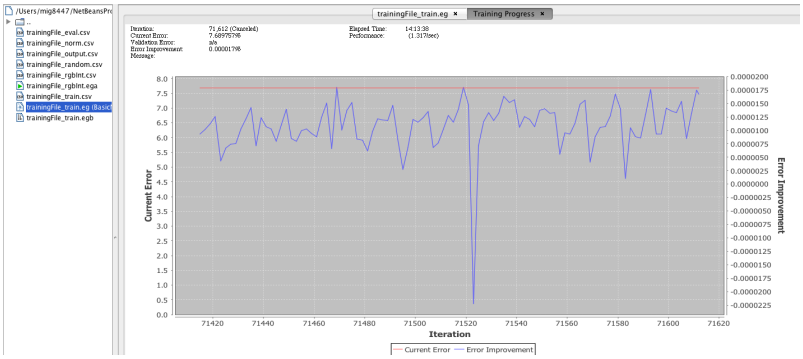


Fig. 5: Feed-forward Neural Network re-trained through Simulated Annealing algorithm with data containing 1 column with all the color channel combined (IntRGB) and 4 categories of Traffic Signs from GTSRB using ENCOG Workbench.

5 Conclusions and Future Work

In this paper we described the development of a method for delivering data to several Machine Learning algorithms in order to compare their performances.

- In the process, we discovered that the pre-processing of the dataset is the most important step for getting accurate results when training any Machine Learning algorithm.
- Speaking of Machine Learning image recognition, pre-processing should deliver the smallest possible amounts of data for describing an image. With an smaller training dataset, the training time and the resultant error will be reduced considerably.
- We also discovered that in order to solve a multi-group classification problem, an artificial Neural Network is a better approach than a Support Vector Machine. This due to the binary nature of the Support Vector Machine.
- After comparing several training techniques, we found that Resilent Propagation was the quickest and most efficient method for training a Neural Network, and that there is a trade off between the training time and the accuracy needed in order to achieve an acceptable result depending on the goal you have.
- We found that for the purpose of testing performance of several Machine Learning algorithms and training techniques, it was a better option to use a renowned Machine Learning Framework instead of implementing our own. This due to the time constraints we had and because we were searching for a Machine Learning algorithm reference.

Some of our ideas for enhancing this work are mostly related to the dataset preprocessing.

- The first of them would be trying to reduce the amount of data sent to the algorithms by using several image processing techniques like converting the image to grayscale or black and white taking the amount of pixels of certain color into account.
- Other idea, that could be used in combination with the first one, is trying to optimize the image brightness and contrast settings in order to normalize the image pixels intensity values.
- The third idea, which again could be used in combination with the previous ones, is to implement a Feature Extraction library including the algorithms mentioned in this paper and apply them to the normalized dataset to get much less data as input for the Machine Learning Algorithms.

References

1. Corina Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Kluwer Academic Publishers* 20, 273-297.

2. Anil K. Jain and Jianchang Mao. 1996. Artificial Neural Networks: A Tutorial. *Theme Feature*.
3. Institut fr Neuroinformatik. 1999. The German Traffic Sign Recognition Benchmark. http://www.rsc.org/dose/titleofsubordinatedocument.Cited27Nov2013.
4. NetPBM. 2013. PPM Format. http://http://netpbm.sourceforge.net/doc/ppm.html.Cited27Nov2013.
5. Heaton Research. 2013. ENCOG Artificial Intelligence Framework for Java. https://code.google.com/p/encog-java/downloads/list.Cited27Nov2013.
6. Brown, Matthew; Szeliski, Richard; Winder Simon. 2004. Multi-Image Matching using Multi-Scale Oriented Patches. http://research.microsoft.com/pubs/70120/tr-2004-133.pdf.Cited26Nov2013
7. Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *Pragmatics* 2:91–110
8. Jeff Heaton. Heaton Research. OCR Applet. http://www.heatonresearch.com/articles/42/page1.html.Cited26Nov2013
9. Egmont-Petersen Michael. IMAGE RECOGNITION WITH NEURAL NETWORKS HOWTO. http://neuroph.sourceforge.net/image_recognition.html.Cited26Nov2013
10. http://code.google.com/p/java-image-scaling/wiki/Getting_started.Cited26Nov2013

Traffic Sign Classification Problem

Iván Michelle García Domínguez

Abstract La presente investigación presenta la propuesta de *Support Vector Machines to Classify Traffic Sign*, esta relacionado con algoritmos de aprendizaje supervisados que analizan y reconocen patrones de datos, usados para clasificación y análisis de regresión. Para este proyecto se usa el Histograma de Gradientes para extraer características de las imágenes y entrenar a nuestro Support Vector Machine, con este algoritmo se obtuvo una buena claificación de las imágenes.

1 Objetivo

El objetivo del presente trabajo de investigación es presentar el método *Support Vector Machines*, que reconoce señales de transito del *German Traffic Sign Recognition Benchmark* y las clasifica.

Durante el semestre se estudiaron diferentes métodos que se usan para que agentes tomen decisiones racionales, este trabajo presenta como se aplican estas técnicas a problemas de la vida real como asistencia al manejar, reconocimiento de escenas, manejo automatizado y que sirven de líneas de investigación para mejorar lo existente en el campo de *Computer Vision*.

2 Introducción

Debido a los avances de la tecnología es cada vez más común encontrar agentes que realizan tareas que antes estaban dedicadas solamente a las personas, tareas que implican un grado de inteligencia, uso de la razón para efectuarlas.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201, e-mail: A00353447@itesm.mx

Una de estas ramas tecnológicas es la “Computer Vision” , donde los agentes son capaces de reconocer figuras y actuar en consecuencia, uno de los algoritmos que resuelve el problema de la clasificación es el tratado en esta investigación.

El reconocimiento de seales de transito puede formar parte del desarrollo de los sistemas inteligentes de transporte o funcionar de forma autnoma; la importancia de estos algoritmos se da por la compleja escena que se da en las ciudades, donde discriminar las seales de transito del área circundante es una tarea nada sencilla, por los que con estas técnicas de busca resolver el problema.

3 Definición del Problema

Las señales de trafico fueron diseñadas para proporcionar información útil a las personas y son fácilmente reconocidas sin ningún problema, estas señales están compuestas de una amplia variedad, por lo que esto dificulta a los sistemas informáticos para clasificarlas. Este es un problema de reconocimiento de patrones.

Lo que se busca es crear un método que reconozca las imágenes y las clasifique, se usaran imágenes de la base de datos *The German Traffic Sign Recognition Benchmark* con el método *Support Vector Machines*.

Estas son las características de las imágenes:
Image format

- The images contain one traffic sign each.
- Images contain a border of 10 % around the actual traffic sign (at least 5 pixels) to allow for edge-based approaches.
- Images are stored in PPM format $PortablePixmap, P6$.
- Image sizes vary between 15x15 to 250x250 pixels.
- Images are not necessarily squared The actual traffic sign is not necessarily centered within the image. . . .

4 Antecedentes

a continuación se presenta el proceso que se emplea para el desarrollo del algoritmo:

Nuestro clasificador depende de OpenCV SVM y de las características funcionales del extractor. se escogi usar el histograma de gradiente orientado (HOG) porque esta visto que proporciona buenos resultados que otros extractores [2]. HOG es capaz de de describir una imagen basado en la distribución local de los gradientes de intensidad. Esta información es combinada y se crea un vector que describe a la imagen.

Antes del proceso del clasificador, el SVM (Support Vector Machine) de ahora en adelante usaremos esta abreviación. debe ser entrenado, y esto se lleva a cabo alimentndolo con imágenes de prueba. Cada vector esta asociado con una clasificación

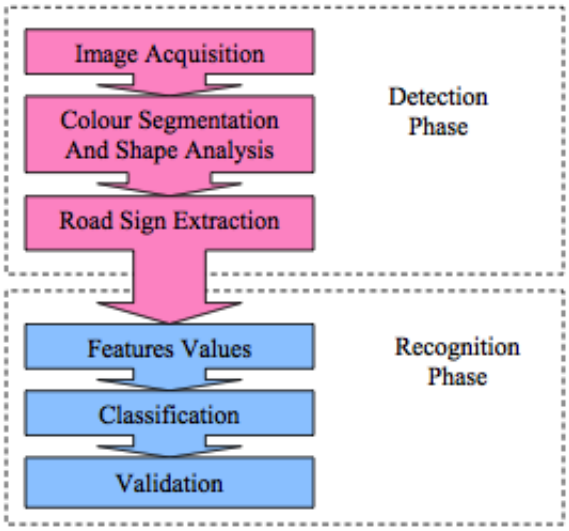


Fig. 1: Proceso de Imagen.

en particular. En el SVM, grupos de vectores de la misma clase son separdos de las otras clases por un hiperplano[3]. Durante la clasificación, HOG es usado para extraer un vector con las caractersticas de la imagen. El vector alimenta la función de predicción. El SVM pone el vector dentro del espacio de vectores existentes y la clase cae dentro de la salida como la entrada de la imagen.

5 Diseño

5.1 Entrenamiento del SVM

Para entrenar al clasificador se le alimenta con el conjunto de imágenes proporcionadas para este caso, se seleccionan las imágenes manualmente y se agrupan por características comunes. se mueven a un directorio definido para el proyecto.

5.2 Extracción de Características

Los archivos *extractor.h / extractor.cpp* definen la interfaz que todos los extractores deben de implementar. El extractor *extractor.cpp* define Dos mťodos que son comunes, el Primero $load_{i}mages$ guarda los archivos de imgenes guardados en el

directorio y las guarda en una matriz del OpenCV. El segundo método $train_SVM$ entrena al SVM automáticamente determinando los parámetros óptimos o usando parámetros predefinidos.

También se declara mtodos cabecera que cada nuevo extractor debe implementar. El método extractor debe adquirir las características de cada imagen. El de clasificación toma las características y las envía la predicción del SVM. El *train* carga los nombres de los archivos de las imágenes y entrena al SVM usando las sus características. El *test* valida si el SVM predice al llamar al método *classify*.

5.3 Extracción usando el HOG

Los archivos *hog_extractor.h / hog_extractor.cpp* definen el extractor HOG, esta clase hereda de la clase extractor y usa el OpenCV HOGDescriptor para computar el vector caracteristico de la imagen. Cuando se extrae de la imagen se reajusta el tamao a 50*x*50 y se computa el vector con un stride de 50*x*50 y el padding de 50*x*50 produciendo un vector de 7,560 atributos.

PSEUDOCODIGO:

Algorithm 2 Algoritmo para extraer la característica de una imagen

```
Require: MAT extract (Mat IMAGE) :
  if IMAGE not empty then
     resize IMAGE to 50x50;
     FEATURES := compute (IMAGE, Size, (50,50), Size(50,50)); return FEATURES;
  end ifreturn empty Mat;
```

Algorithm 3 Extracción de características de un conjunto de imgenes almacenamiento

```
Void extract_features (string PATH, vector <string> CLASSES, vector <vector <string > >
NAMES: )
for each C in CLASSES: do
    for each N in NAMES: do
        FULL_PATH := PATH + CLASSES[C] + "/" + NAMES[C] [N];
        IMAGE := imread(FULL_PATH);
        FEATURES := extract(IMAGE);
        associate FEATURES with CLASSES[C] and store;
    end for
end for
```

Algorithm 4 Extracción de las características de una imagen y clasificación usando un SVM existente

```
int classify (Mat IMAGE) :
if SVM is not initialized : then
    exit;
end if
FEATURES := extract (IMAGE);
TYPE := SVM - >predict (FEATURES); return TYPE;
```

Algorithm 5 Cargar un conjunto de imágenes, extraer sus características y entrenar el SVM

```
void train ( string TRAIN_PATH, string SAVE_PATH ) :
    CLASSES, NAMES := load_images (TRAIN_PATH);
    extract_features (TRAIN_PATH, CLASSES, NAMES ) ;
    train_SVM (SAVE_PATH);
```

Algorithm 6 Cargar un conjunto de imágenes y clasificarlas usando un SVM

```
Void test (string TEST_PATH) :
    CLASSES, NAMES := load_images (TEST_PATH) ;
for each C in CLASSES : do
    for each N in NAMES: do
        FULL_PATH := PATH + CLASSES[C] + "/" + NAMES[C] [N];
        IMAGE := imread(FULL_PATH);
        TYPE := classify (IMAGE) ;
        if TYPE == CLASSES[C] : then
            predictioniscorrect;;
        end if
    end for
end for
```

5.4 Clasificación de imagenes

El archivo *ml_classify.cpp* define un nodo que carga un SVM y clasifica la imagen, la guarda en un subdirectorio de a cuerdo a la clasificación de la imagen. esto permite a los usuarios ver como fue clasificada cada imagen.

5.5 Metodología de Prueba

Se uso el archivo *ml_run.cpp* para llevar a cabo las pruebas estadísticas y de desempeño en nuestro SVM y la implementación del HOG.

se definió que el nṁuero de Imagenes de prueba seria de 10% del número de Imagenes de Entranmiento. A continuación una tabla representativa:

Por cada muestra se corrieron 40 pruebas, al azar se seleccionó el conjunto de imágenes de prueba y entrenamiento.

Table 1: Tamaño de Muestras

Training Images	Testing Images	Total Images
78	7	85
156	15	171
312	31	343
624	62	686
1248	124	1372
2496	249	2745

6 Resultados

Los resultados muestran que cuando el número de imágenes aumenta, la precisión y el tiempo para clasificar una imagen incrementa. Miemntras el número de pruebas por muestra aumenta la tasa de precisión baja. La gráfica abajo sugiere que a cierto número de del tamaño de la muestra no vemos incremento en la predicción de imgenes. Y como se cuenta con un considerable número de imágenes no podemos determinar el tamaño optimo de de imágenes para entrenar al SVM. Se piensa que si se usa una vector con más características no nos da una mejora en los resultados y el costo de examinar estas características es mayor.

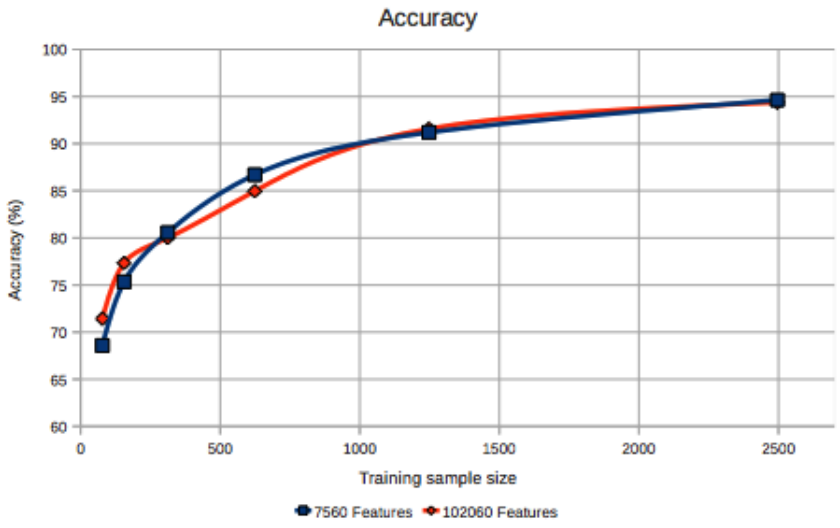


Fig. 2: Resultado de Precisión.

Cuando el número de características usadas para entrenar desde 7560 a 102,060 el tiempo para clasificar una imagen aumenta dramáticamente.

Fig. 3: Resultado del Tiempo al clasificar.

Con las siguientes gráficas no se ve una diferencia considerable en el nṁuero de falsos positivos y falsos negativos, entre los vectores de las características. De esto podemos inferir que no es necesario aumentar el número de características para una mejor respuesta del SVM.

7 Conclusiones

Se obtuvo un sistema con una precisión considerable usando “Support Vector Machines”. Una herramienta poderosa para la clasificación de señales de transito o cualquier figura apoyado en el Histograma.

Los “Support Vector Machines” fueron introducidos por Vapnik en 1992, y han ganado atención gracias a un gran número de merits teóricos y computacionales. Sus raices están en la teoría de aprendizaje usando estadísticas . Para resolver un problema de clasificación, el SVM separa los datos en categorías.

Este sistema, involucra una mezcla de “Computer Vision” y reconocimiento de patrones. Esta investigación abre una nueva frontera para investigaciones relacionadas con sistema inteligentes en el futuro. Puede ser parte de un proyecto automatizado de reconocimiento de señales manejando un automóvil.

Acknowledgements Agradezco a los profesores, me mostraron un nuevo camino donde puedo desarrollar nuevas habilidades y aprender nuevas cosas.

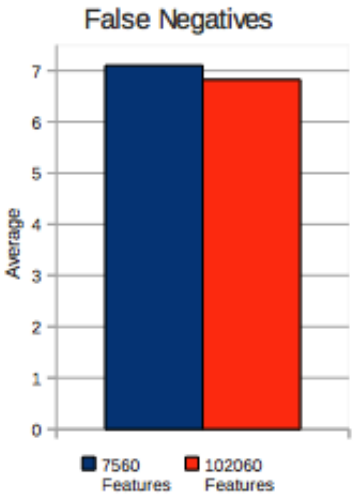


Fig. 4: Promedio de Falsos Negativos por tamaño de Vector.

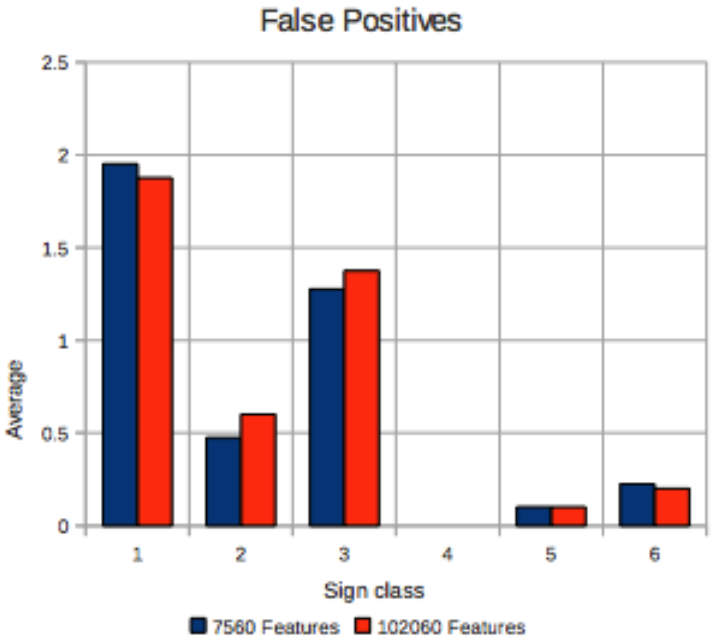


Fig. 5: Promedio por Falsos Positivos por características de Vectores.

References

[1] Dalal, N. and Triggs, B., “Histograms of Oriented Gradients for Human Detection”, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 886 ? 893, 2005.

[2] Beeson, P., O’Quin, J., Gillan, B., Nimmagadda, T., Ristroph, M., Li, D., Stone, P., “Multiagent Interactions in Urban Driving”, Journal of Physical Agents: Multi-Robot Systems, vol. 2, no. 1, 2008.

[3] Wikipedia contributors, “Support vector machine”, Wikipedia, The Free Encyclopedia, en.wikipedia.org/wiki/Support_vector_machine, 5 May. 2011.

[4] Breckon, T., 9 Feb. 2011, “Support Vector Machine (SVM) learning”, ver. 0.2, [C++ Program], public.cranfield.ac.uk/c5354/teaching/ml/examples/c++/speech_ex/svm.cpp, 29 Mar. 2011.

Comparative for Feature Extraction between Zernike Moments using a Multilayer Perceptron Classifier and The German Traffic Sign Benchmark

Julio Cesar Roa Gil

Abstract The recognition of traffic signs are a difficult activity for any computer system. The signs on the roads were designed to be identified by drivers easily. However, weather factors affects the the recognition by computers, which in humans is not as noticeable. Artificial neural networks can be used to recognize patterns from the moments obtained by traffic sign imaging and classify them as true or false. This can be done using Artificial Neural Networks for pattern recognition using a feed-forward four-layered neural network trained with a backpropagation algorithm. A comparison between Zernike Moments method and Histogram of Oriented Gradient (HOG) features to classify traffic signs based on The German Traffic Sign Benchmark was done. The images were pre-processed: resizing and extracting the Zernike moments. The neural network was trained using a part of the dataset and then tested with the remaining dataset obtaining a classification accuracy of 86.84% and the best accuracy reported by the Institut für Neuroinformatik for HOG features was 95.68% [**Stallkamp-IJCNN-2011**]. Comparing these two results, we can concluded that the Zernike moments, used as feature extraction, can achieve very precise levels for imaging classification.

1 Introduction

The recognition of traffic signs in the field of image recognition is having a boom thanks to the use of different artificial intelligence techniques such as neural networks and support vector machines. Actually it is a real-world challenge on computer vision field.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201. ee-mail: roa.juliocesar@gmail.com

The signs on the roads were designed to be identified by drivers easily because it follows a standard in color, text and size. Different factors such as climate, light intensity, scale, etc. greatly affect the recognition by computers, which in humans is not as noticeable.

The main objective is to compare the results obtained by Institut für Neuroinformatik in his project The German Traffic Sign Benchmark by implementing a neural network and a pre-processing step the images through feature extraction as shape and location.

In this work, we consider a statistical approach to image recognition using Zernike moments as features. Zernike moments have been used in the optical character recognition and image recognition communities with good results[**zernike˙results**]. The rotation invariance property of Zernike moments is especially desirable for symbol recognition.

The scope is determined to: Perform a neural network, specifically multilayer perceptron (MLP) that allows identification of six types of signs. The speed limit signs (20 km, 30 km, 50 km, 60 km, 70 and 80 km) specifically. As a first step it has been performed pre-processing the image to develop uniformity of sizes of all the images of each of the classes.

For this goal was used the bilinear interpolation algorithm, after that the method of Zernike Moments was applied. Finally, once the experiments have been done to compare the results obtained with results obtained by the Institut für Neuroinformatik. In summary, comparing the method of Zernike Moments vs HOG (Histogram of Oriented Gradients).

The paper is organized as follows: Section 2 gives a description of our target traffic sign set. A previous work are described in Section 3. Image preprocessing, feature extraction, and classification methods are presented in subsection 4.1, 4.2, and 4.3. Results are presented in subsection 4.4 and the conclusions in the Section 5.

2 Problem Definition

Through the implementation of a multilayer neural network (multilayer perceptron) are going to be identified and classify a subset of traffic signals from The German Traffic Sign Benchmark, this subset is described in the following table:

As evidenced in the table is taken into account 6 different types of traffic signals (signals speed limit), which is a subset of the database which contains 43 kinds of signals class and around 50,000 images. This problem also involves the implemen-

Table 1: Scope and matched class id with traffic sign

Class ID	Traffic Sign
00000	Speed limit 20 Km
00001	Speed limit 30 Km
00002	Speed limit 50 Km
00003	Speed limit 60 Km
00004	Speed limit 70 Km
00005	Speed limit 80 Km

Fig. 1: Speed limit traffic signs

tation of a method to standardize the size of the images, because the images are in range of 15x15 to 250x250 pixels.

For this reason we performed a resize to 60x60 pixels scale on each of the images using bilinear interpolation. Additionally to ensure the best performance of the neural network, is conduct a image processing through a method called Zernike Moments for shape and location characteristics, which are methods that are used for processes feature extraction and classification in image processing.

3 Previous work

Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition[**Stallkamp-IJCNN-2011**]. Traffic signs are characterized by a wide variability in their visual appearance in real-world environments. Both image processing and machine learning algorithms are continuously refined to improve on this task. They present a publicly available traffic sign dataset with more than 50,000 images of German road signs in 43 classes. The data was considered in the second stage of the German Traffic Sign Recognition Benchmark held at IJCNN 2011. In this research, they compare the traffic sign recognition performance of humans to that of state-of-the-art machine learning algorithms. They present the extended GTSRB dataset with 51,840 images of German road signs in 43 classes.

Also, there has been a significant amount of research to date in various aspects of image recognition-based on Zernike Moments: Sketched symbol recognition[**sketched**],

chinese character recognition[**chinese**], fingerprint classification [**fingerprint**].

In the work of Sketched Symbol Recognition, they presented an on-line recognition method for hand-sketched symbols. The method is independent of stroke-order, number, and direction, as well as invariant to rotation, scaling, and translation of symbols. The Zernike moment descriptors are used to represent the symbols. Support Vector Machines (SVM), Minimum Mean Distance (MMD), and Nearest Neighbor (NN) classifiers are trained on 7,410 sketched symbols. It achieved a correct classification rate of 97%.

Kasza[**chinese**] presented a pseudo-Zernike feature descriptor based recognition technique for accurate identification of printed and handwritten Chinese characters. He has shown that using pseudo-Zernike moments for feature descriptors is a viable option for classifying structurally complex images. Moment functions have been successfully applied to many shape recognition problems, due to the fact that they tend to capture global features, which makes them well suited as feature descriptors. They provide excellent invariance features and exhibit better performance than other moment based solutions. Even when the input is stained with heavy amounts of noise, they still provide an accurate method for character identification.

Finally, on the fingerprint classification work, they presented a complete fingerprint recognition system using an Artificial Neural Network (ANN). The ANN was trained by backpropagation algorithm on a dataset of more than 400 fingerprints with 10 samples of each person (40 people). The structure of the ANN was decided experimentally. Pseudo Zernike Moments (PZM) will be used as a features vector for all images. It achieved a correct classification rate of 98%.

4 Experiments

4.1 Experiment Data

A collection of data has been gathered and selected from The German Traffic Sign Recognition Benchmark[**Stallkamp-IJCNN-2011**]: public database GTRSB.

After download the public database, we selected a random sample of 100 images for each of the classes. Then, we splitted these one hundred images, 80% has been used as the training set and the remaining 20% as the test set.

4.2 Preprocessing

In many image-processing applications, digital images must be zoomed to enlarge image details and highlight any small structures present. This is done by making multiple copies of the pixels in a selected region of interest (ROI) within the image. Several algorithms are used to perform such an operation.

Fractionally zoomed images can also be obtained by varying the number of copies made of each pixel in the ROI. An interpolation technique that reduces the visual distortion caused by the fractional zoom calculation is the bilinear interpolation algorithm, where the fractional part of the pixel address is used to compute a weighted average of pixel brightness values over a small neighborhood of pixels in the source image. Bilinear interpolation produces pseudoresolution that gives a more aesthetically pleasing result[**bilinear**].

Zernike moments are not invariant to scale and translation, therefore the traffic signs are first scaled. In order to standardize the size of the images we performed a resize to 60x60 pixels scale on each of the images using bilinear interpolation with PIL library for Python.

Fig. 2 Scaling traffic sign example

After finished the scaling process[**size˙normalization**], it continued with the feature extraction[**zernike˙shape**].

Zernike polynomials are widely used as basis functions of image moments. Since Zernike polynomials are orthogonal to each other, Zernike moments can represent properties of an image with no redundancy or overlap of information between the moments. Although Zernike moments are significantly dependent on the scaling and the translation of the object in an ROI, their magnitudes are independent of the rotation angle of the object. Thus, they can be utilized to extract features from images that describe the shape characteristics of an object.

In summary, Zernike moments are a class of orthogonal moments and have been shown effective interms of image representation. Zernike moments are rotation invariant and can be easily constructed to an arbitrary order. Although higher order moments carry more fine details of an image, they are also more susceptible to noise.

The zernike moments[**zernike˙moments**, **zernike˙moments˙efficient**] was applied to the extract features using Mahotas[**mahotas**] library for Python. The results were several vectors with the relevant information about each one image, exactly 25 data for each image and saved in a single file.

4.3 Learning and classification

Since there is no visible pattern, artificial neural networks were used for pattern recognition in order to classify signals as truth-telling or deception. Neuroph software was used for this purpose. The behavior of a neural network is defined by the way its individual computing elements are connected and by the strengths of those connections, or weights. The weights are automatically adjusted by training the network according to a specified learning rule until it performs the desired task correctly.

In this paper, an MLP neural network is used as a classifier in a traffic sign recognition where the inputs to the neural network are feature vectors derived from the proposed feature extraction technique described in the previous section.

The construction of the MLP neural network involves four different layers; input layer, two hidden layers and output layer, with feed forward architecture. The MLP are fully connected between layers. The network has sigmoid hidden and output neurons. The network is trained with scaled conjugate gradient back propagation.

The number of input neurons in the MLP is the same as the dimension of the Zernike Moments (order 8) and the number of output neurons is 6 since we have 6 classes. The number of hidden neurons is determined experimentally.

The structure of the neural network has twenty five neurons in the input layer, also it has fifteen neurons in the hidden layers and six neurons in the output layer as shown in Fig. 2.

In order to configure the different classes in the output layer, we configured the output for each class as shows in Table 2:

Those outputs are part of the Zernike Moments vector for each class. The output considered each neuron in the output layer as a new field in the vector. For this reason six values was adding in order to train the MLP in a supervised way.

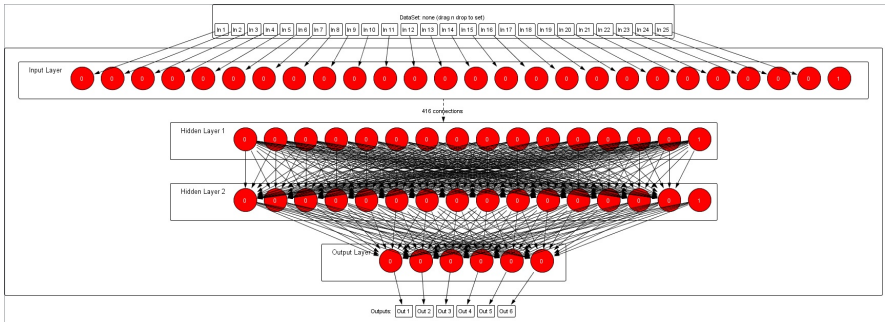


Fig. 3: Mulilayer perceptron structure

Table 2: Output configuration

Class ID	Traffic Sign	Output
00000	Speed limit 20 Km	0 0 0 0 0 0
00001	Speed limit 30 Km	0 1 0 0 0 0
00002	Speed limit 50 Km	0 1 1 0 0 0
00003	Speed limit 60 Km	0 1 1 1 0 0
00004	Speed limit 70 Km	0 1 1 1 1 0
00005	Speed limit 80 Km	0 1 1 1 1 1

4.4 Results

After trainned with several configurations and architectures as shown in Table 2, we achieved an accuracy of 99.1% on training phase as shown in Figures 4 and 5 with the structure shown in Fig. 2. Finally in the testing phase, we obtained an accuracy of 86.84%.

Table 3, shows various examples of results on the testing phase. It shows up the MLP output and the desired output and the MLP error.

The following figures shows up the network error (Figure 4,5), weight histogram (Figure 6) and surface graphs (Figure 7).

5 Conclusions

We have developed a Multilayer Perceptron Classifier that is invariant to rotation and translation. A recognition accuracy rate of 86.84% had been obtained using

Table 3: Experiment result using 15 hidden neurons and different configurations

Momentum	Learning rate	Error
0,9	0,1	45,35%
0,9	0,2	46,63%
0,9	0,3	43,27%
0,9	0,4	47,72%
0,9	0,5	48,28%
0,8	0,1	39,90%
0,8	0,2	40,46%
0,8	0,3	38,05%
0,8	0,4	40,54%
0,8	0,5	40,24%
0,7	**0,1**	**0,09**%
0,7	0,2	37,68%
0,7	0,3	37,38%
0,7	0,4	38,21%
0,7	0,5	39,15%
0,6	0,1	36,06%
0,6	0,2	37,08%
0,6	0,3	38,77%
0,6	0,4	38,28%
0,6	0,5	50,03%
0,5	0,1	48,18%
0,5	0,2	48,93%
0,5	0,3	50,73%
0,5	0,4	51,86%
0,5	0,5	52,63%

Table 4: Example of MLP testing results

MLP output	Desired output	MLP Error
0; 1; 1; 1; 1; 1	0; 1; 1; 1; 1; 1;	0; -0; -0; -0; -0; -0;
0; 1; 1; 1; 1; 0;	0; 1; 1; 1; 0; 0;	0; -0; -0; -0; 1; 0;
0; 1; 1; 1; 1; 0;	0; 1; 1; 1; 0; 0;	0; -0; -0; -0; 1; 0;
0; 1; 1; 1; 1; 0;	0; 1; 1; 1; 0; 0;	0; -0; -0; -0; 0,9885; 0;
0; 1; 1; 1; 1; 0,0255;	0; 1; 0; 0; 0; 0;	0; -0; 1; 1; 1; 0,0255;
0; 0; 0; 0; 0; 0;	0; 0; 0; 0; 0; 0;	0; 0; 0; 0; 0; 0;
0; 1; 1; 1; 1; 0;	0; 1; 1; 1; 0; 0;	0; -0; -0; -0; 1; 0;
0; 0,9978; 0,9885; 0,0011; 0,0005; 0;	0; 1; 1; 0; 0; 0;	0; -0,0022; -0,0115; 0,0011; 0,0005; 0;
0; 0,9976; 0,9999; 1; 0,9947; 0,957;	0; 0; 0; 0; 0; 0;	0; 0,9976; 0,9999; 1; 0,9947; 0,957;
0; 0,9978; 0,9885; 0,0011; 0,0005; 0;	0; 1; 1; 0; 0; 0;	0; -0,0022; -0,0115; 0,0011; 0,0005; 0;
0; 0,9631; 0,0005; 0,0004; 0,0012; 0,0003;	0; 1; 1; 0; 0; 0;	0; -0,0369; -0,9995; 0,0004; 0,0012; 0,0003;
0; 1; 1; 1; 0,9684; 0;	0; 1; 1; 1; 0; 0;	0; -0; -0; -0; 0,9684; 0;

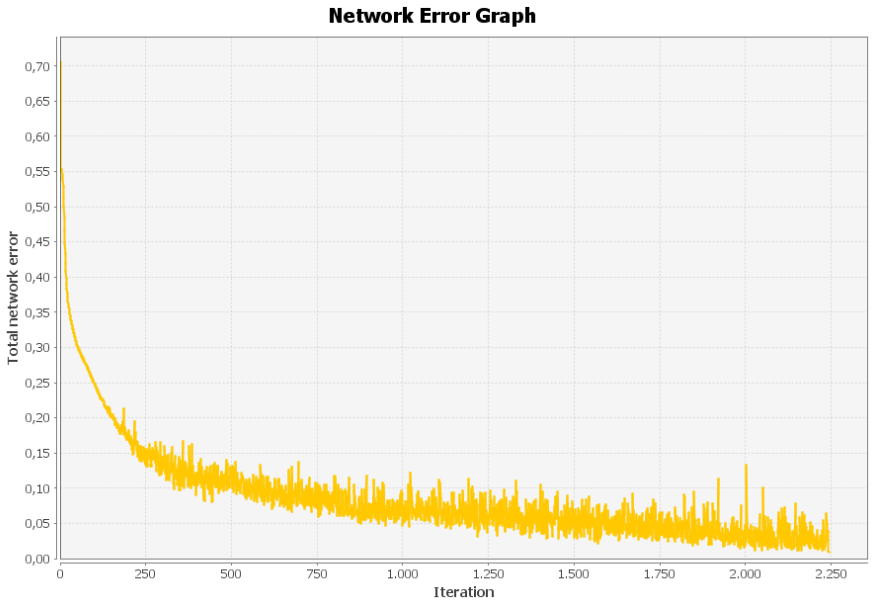


Fig. 4: Training phase - Network error

Zernike moments up to order 8 as feature extraction. It is an excellent accuracy just for a few dataset and a medium Zernike's order.

Comparing the results between the Zernike moments (86.84%) and HOG (95.68%)[**Stallkamp-IJCNN-2011**], we conclude that the Zernike moments, used as feature extraction, can achieve very precise levels for image representation and combined with MLP, it is a good classifier for the traffic signs. Moreover, in a general way, the combination between Zernike Moments as feature extraction phase and MLP as classifier achieved excellent outcomes on recognition image process.

References

1. Stallkamp, J.; Schlipsing, M.; Salmen, J.; Igel, C., "The German Traffic Sign Recognition Benchmark: A multi-class classification competition," Neural Networks (IJCNN), The 2011 International Joint Conference on , vol., no., pp.1453,1460, July 31 2011-Aug. 5 2011 doi: 10.1109/IJCNN.2011.6033395
2. He, Chun Lei, Ping Zhang, Jian Xiong Dong, Ching Y. Suen, and Tien D. Bui. The Role of Size Normalization on the Recognition Rate of Hanwritten Numerals. In The 1st IAPR TC3 NNLPAR Workshop, 812, 2005.
   `http://www.dsi.unifi.it/NNLDAR/Papers/02-NNLDAR05-zhang.pdf`
3. Lin, Tsong-Wuu, & Yun-Feng Chou. A Comparative Study of Zernike Moments. In Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference. **169**, 516–519, (2003).

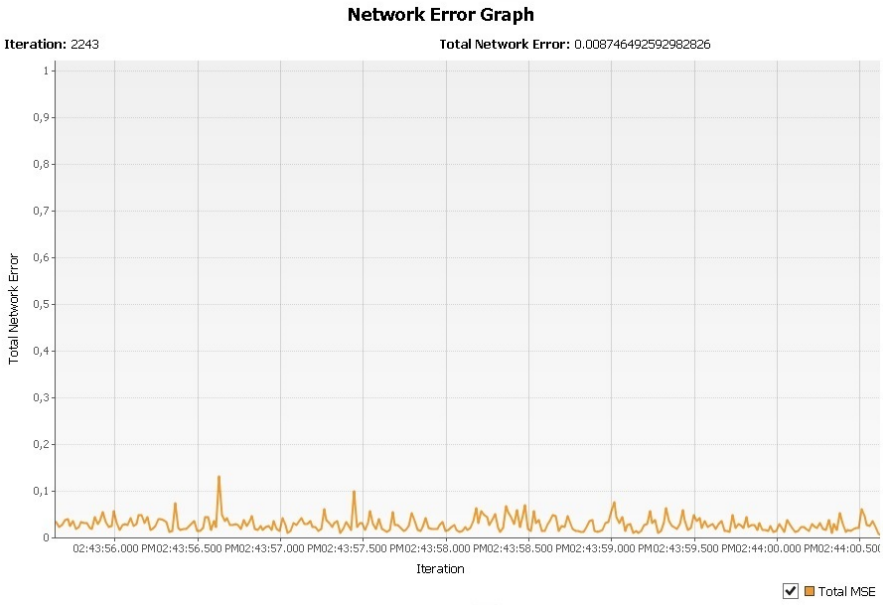


Fig. 5: Training phase - Real time network error

4. Khotanzad, A., & Hong, Y. H. (1990). Invariant image recognition by Zernike moments. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 12(5), 489497.
5. Vorobyov, M. (2011). Shape Classification Using Zernike Moments.
6. Papakostas, G. A., Y. S. Boutalis, D. A. Karras, and B. G. Mertzios. Efficient Computation of Zernike and Pseudo-Zernike Moments for Pattern Classification Applications. Pattern Recognition and Image Analysis 20, no. 1 (April 6, 2010): 5664. doi: 10.1134/S1054661810010050.
7. Coelho, L.P. 2013. Mahotas: Open source software for scriptable computer vision. Journal of Open Research Software 1(1):e3, DOI: http://dx.doi.org/10.5334/jors.ac
8. Hse, H.; Newton, A.R., "Sketched symbol recognition using Zernike moments," Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on , vol.1, no., pp.367,370 Vol.1, 23-26 Aug. 2004 doi: 10.1109/ICPR.2004.1334128
9. Kasza, P. (n.d.). Pseudo-Zernike Moments for Feature Extraction and Chinese Character Recognition.
10. El-Feghi, I.; Tahar, A.; Ahmadi, M., "Efficient features extraction for fingerprint classification with multi layer perceptron neural network," Signals, Circuits and Systems (ISSCS), 2011 10th International Symposium on , vol., no., pp.1,4, June 30 2011-July 1 2011 doi: 10.1109/ISSCS.2011.5978683
11. Editorial. (n.d.). UNDERSTANDING IMAGE-INTERPOLATION TECHNIQUES. Retrieved from http://www.vision-systems.com/articles/print/volume-12/issue-10/departments/wilsons-websites/understanding-image-interpolation-techniques.html

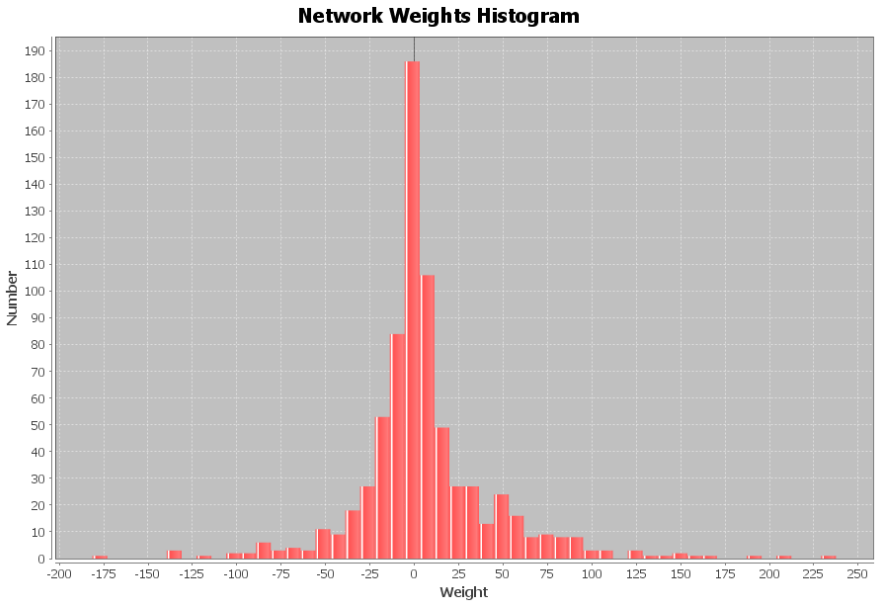


Fig. 6: Network weights histogram

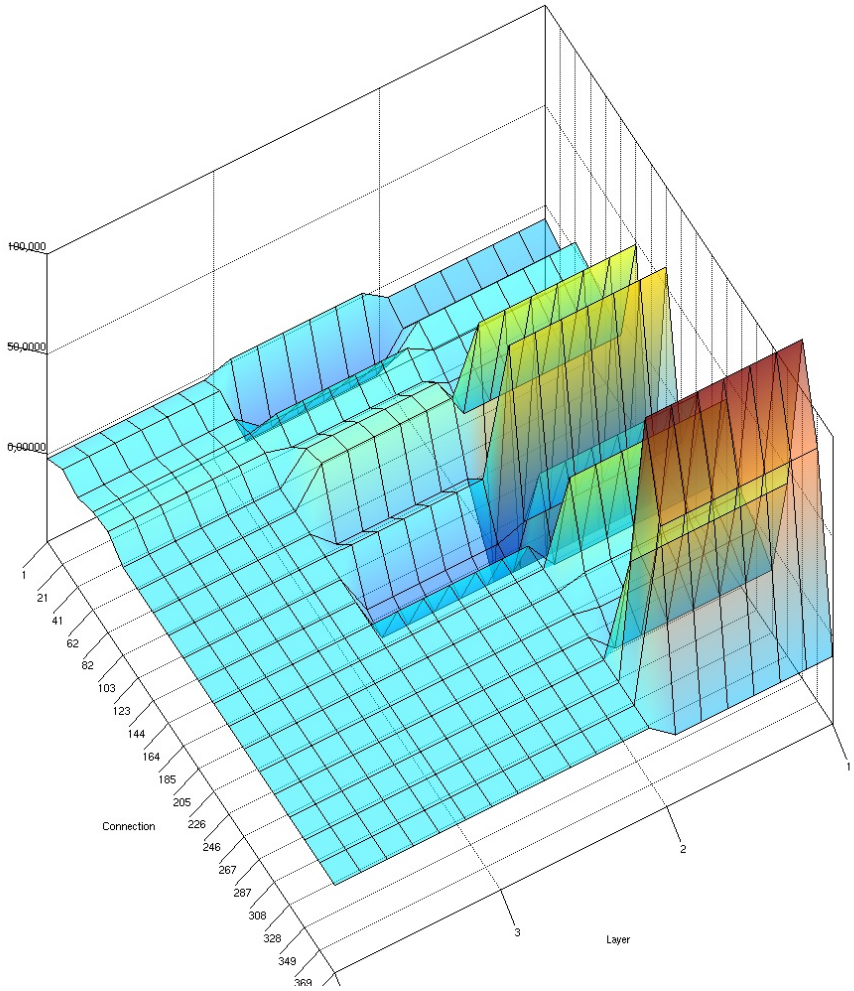


Fig. 7: Network weights surface 3D graph

Chapter 2
SPAM Detection

Naive Bayesian classifiers for detecting SPAM emails & SMSs

Alvaro Maza

Abstract Spam is information produced to be delivered to a large number of recipients, regardless of their wishes. This paper discusses the current validity of Naive Bayesian classifiers method and illustrates its current effectiveness to detect spam emails and SMSs.

Gary Robinson proposed a Bayesian Statistical approach to detect spam emails [**Robinson-2003**]. Naive Bayesian Classifiers work by correlating the use of tokens (i.e. words), with spam and non-spam emails and then using Bayesian, inference to calculate a probability whether or not it is spam. One of its key benefits is that it is self-adapting.

The current effectiveness of the classifiers was measured by comparing the number of right and wrong classifications done against one public dataset of 2002 of spam and non-spam emails, and one recent dataset of 2010. Additionally, we measured the effectiveness of the classifiers against a recent SMS dataset of 2012.

Results suggest that Naive Bayesian classifiers are still very accurate in detecting spam emails as only few false negative results were obtained — which are generally acceptable to users. Finally, we found that Naive Bayesian classifiers can also be used to detect spam SMSs.

1 Introduction

1.1 Problem

The Spam Track at the Text Retrieval Conference (TREC) defines email spam as [**Cormack-2005**]:

> Unsolicited, unwanted email that was sent indiscriminately, directly or indirectly, by a sender having no current relationship with the recipient.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201. e-mail: a00354420@itesm.mx

The motivation behind spam is to have information delivered to the recipient that contains a payload such as advertising for a product, bait for a fraud scheme, promotion of a cause, or computer malware designed to hijack the recipients computer. Because it is so cheap to send information, only a very small fraction of targeted recipients need to receive and respond to the payload for spam to be profitable to its sender .

Spam is more than just a minor problem. Every day around 100 billion emails are sent to valid email addresses around the world; in 2010, 88% of the email traffic was spam [**Symantec-2010**, **MAAWG-2011**]. It is estimated that 120-billion junk emails will be sent every day within the next four years .

According to a study conducted by Nucleus Research Inc., spam management costs U.S. businesses more than $71 billion annually in decreased productivity as well as in technical expenses — $712 per employee [**Nucleus-2003**]. Predictions for the future costs of spam dont look any brighter. Indeed, an employee can inadvertently reply to a spam email and potentially deliver information to someone who should not have received that information.

1.2 Background

Paul Graham [**Graham-2002**], author of several books on Lisp, suggested an approach to filter spam in his article, "A Plan for Spam". After that, Gary Robinson took his approach for generating probabilities associated with words and proposed a Bayesian calculation for dealing with words that hadn't appeared very often [**Robinson-2003**]. Then he suggested an approach based on the chi-square distribution for combining the individual word probabilities into a combined probability representing an email [**Robinson-2004**]. Finally, Tim Peters of the Spambayes Project proposed a way of generating a particularly useful spamminess indicator based on the combined probabilities [**Peters-2004**].

We took those papers and develop a tool to test their current validity and effectiveness againsts predefined datasets of emails and SMSs.

Other methods (e.g. Ad Hoc Classifiers, Rule-Based Filtering, etc) and other recent learning models discussed by Gordon Cormack [**Cormack-2006**] seem also efficient but are not discussed in this paper.

2 Methodology

2.1 Nave Bayesian Classifiers

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong independence assumptions (naive). In simple terms, a naive

Bayes classifier assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 3" in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of the presence or absence of the other features.

An advantage of the naive Bayes classifier is that it only requires a small amount of training data to estimate the parameters necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix.

Abstractly, the probability model for a classifier is a conditional model.

$$p(C|F_1, F_2, ..., F_n) \tag{1}$$

over a dependent class variable C with a small number of outcomes or classes, conditional on several feature variables F_1 through F_n. Using Bayes' theorem, equation 1 can be written as

$$p(C|F_1, F_2, ..., F_n) = \frac{p(C)p(F_1, ..., F_n|C)}{p(F_1, ..., F_n)} \tag{2}$$

Bayes' rule relates the probability of a binary condition C, given a positive result W of a test for C. If $P(x|y)$ is read as "the probability of x given y", and C' is the complement of C, Bayes' rule can be simplified as

$$p(C|W) = \frac{p(W|C) * P(C)}{P(W|C) * P(C) + P(W|C') * P(C')} \tag{3}$$

2.2 Implementation Details

Nave Bayesian method is used for the learning process. Each email is analyzed to calculate its probability of being spam or not using individual characteristic of words in the email.

Let's express each of the terms of the equation 3 in spam-filtering language. C is the condition "the message is spam". W is "the word w is present".

- $P(C|W)$ can be interpreted as the probability (p) that a randomly chosen email containing the word w is Spam.

$$p(C|W) = p \tag{4}$$

- $P(W|C)$ is the probability that a spam will contain the word w. Our estimate of that is the proportion of spam messages used to build the training database.

$$p(W|C) = \frac{number\ of\ spam\ emails\ containing\ the\ word\ w}{total\ number\ of\ spam\ emails} \tag{5}$$

- $P(C)$ is the probability that a random message is spam. We are making the assumption that there is no a priori reason for any incoming message to be spam rather than non-spam, and we consider both cases to have equal probabilities.

$$p(C) = 0.5 \tag{6}$$

- $P(W|C')$ is the probability that a non-spam will contain the word w. Our estimate of that is the proportion of non-spam messages used to build the training database.

$$p(W|C') = \frac{number\ of\ non-spam\ emails\ containing\ the\ word\ w}{total\ number\ of\ non-spam\ emails} \tag{7}$$

- $P(C')$ is the probability that a random message is not spam. As stated before, we consider both cases to have equal probabilities.

$$p(C') = 0.5 \tag{8}$$

Given the above definitions, we can see that equation 3, can be simplified as:

$$p(C|W) = \frac{p(W|C)}{P(W|C) + P(W|C')} \tag{9}$$

The tool implementing the Bayesian algorithm first needs to learn from a predefined set of various spam and non-spam emails (training set). Then, it needs to parse each email to extract the words of interest and finally implement the Naive Bayesian Method, described above, to be able to classify other new emails. The tool can compute p(w) for every word in an email and use that information as the basis for further calculations to determine whether the email is spam or not.

In order to proper parse the words of interest in the emails parts and be able to form tokens, some common words shall be removed (e.g. you, I, for, is, are, etc). If the email body has any HTML tag, it must be stripped. Additionally, if a word appears more than one time in a single email, the count should not be increased.

2.2.1 Degree of belief

There is a problem with probabilities calculated as above when words are very rare. For instance, if a word appears in exactly one email, and it is a spam, the calculated p(w) is 1.0. But clearly it is not absolutely certain that all future emails containing that word will be spam; in fact, we simply don't have enough data to know the real probability.

Gary Robinson's proposal [**Robinson-2003**] is used to calculate the degree of belief of a word w:

$$f(w) = \frac{(s*x) + (n*p(w))}{s+n} \tag{10}$$

Where:

$$s = Assumed\ strength\ of\ the\ background\ information. \tag{11}$$

$$x = Assumed\ probability\ of\ the\ background\ information. \tag{12}$$

$$n = Number\ of\ emails\ received\ containing\ word\ w. \tag{13}$$

This gives us the convenient use of x to represent our assumed probability from background information and s as the strength we will give that assumption. We set reasonable values to s to 1 and x to 0.5. In practice, the values for s and x are found through testing to optimize performance.

2.2.2 Combining the Probabilities

At this point, we can compute a probability, f(w), for each word that appears in a new email. This probability reflects our degree of belief, based on our background knowledge and on data in our training, that an email chosen randomly from those that contain w will be a spam.

So each email is represented by a set of probabilities and combining these individual probabilities can give us the overall indicator. We use Fisher's Method to combine the probabilities:

$$H = \chi^{-1}(-2\ln(\prod(f(w)), 2n) \tag{14}$$

$$S = \chi^{-1}(-2\ln(\prod(1-f(w)), 2n) \tag{15}$$

where χ^{-1} is the inverse chi-square function, used to derive a value from a chi-square-distributed random variable.

Finally, we can calculate our indicator of spamminess (Peters, 2004):

$$I = \frac{1+H-S}{2} \tag{16}$$

3 Experiments and Results

As mentioned earlier, in this paper we will concentrate on verifying the current validity of the Naive Bayes Classifiers in identifying spam emails. In order to do that, we compared the effectiveness againts two different datasets. The first dataset contains 5686 emails sent in 2002 and the second dataset contains 4312 emails sent in 2010. Both datasets are originally in english language.

Additionally we run the same Naive Bayesian Classifiers method against a dataset of 5573 SMSs sent in 2012. Results obtained are shown below.

3.1 SpamAssassin emails dataset - 2002

The first dataset was obtained from The Apache SpamAssassin Project (2002) [**SpamAssassin-2002**]. It contains 3300 non-spam emails and 2386 spam emails, from which 1592 non-spam and 1189 spam emails were randomly chosen to train the tool. The rest of the emails were processed by the tool and the results are shown in tables 1, 2 and 3.

The Naive Bayesian Classifiers method was idenpendently executed on each of the main parts of an email — the subject (table 1), the body (table 2) and the email sender (table 3). It is shown the percentage of spam emails considering different thresholds for the indicator of spamminess.

Table 1: Naive Bayesian Classifiers applied on the subjects - SpamAssassin Dataset 2002

Set of Testing Emails	% SPAM Isubject > 0.50	% SPAM Isubject > 0.60	% SPAM Isubject > 0.70	% SPAM Isubject > 0.80	% SPAM Isubject > 0.90
N. SPAM SET (1708)	1.93%	0.29%	0.18%	0.0%	0.0%
SPAM SET (1197)	75.10%	31.19%	29.49%	21.30%	11.9%

Some highlights considering only the subject of the emails to determine whether an email is spam or not:

- We got a very small rate of false positive results (i.e. 1.93% of the non-spam testing set) and a fair false negative rate (i.e.$\sim$25% of the spam testing) using an indicator of spamminess over 0.50.
- The percentage of emails which obtained the indicator of spamminess over 0.60, dramatically got reduced on both subsets: to 0.29% of the non-spam and to 31.19% of the spam set. This means that most of the emails got an indicator of spamminess between 0.50 and 0.60. Consequently, the subject of the emails by itself can not be used to determine if an email is spam or not. Presumably because since the subjects are usually composed by few words, they can be found on both training sets.
- From that point, as we increase the theshold indicator, the percentage of spam emails decrease in small rates. Those emails represent the obvious spam emails. It is very likely that most of their words are only found in our spam training set.

Table 2: Naive Bayesian Classifiers applied on the bodies - SpamAssassin Dataset 2002

Set of Testing Emails	% SPAM IBody > 0.50	% SPAM IBody > 0.60	% SPAM IBody > 0.70	% SPAM IBody > 0.80	% SPAM IBody > 0.90
N.SPAM SET (1708)	0.64%	0.0%	0.0%	0.0%	0.0%
SPAM SET (1197)	75.27%	40.69%	35.92%	31.62%	25.65%

Some highlights considering only the body of the emails to determine whether it is spam or not:

- Again, we got a very small rate of false positive results (i.e. 0.64% of the non-spam testing set) and a fair false negative rate (i.e. $\sim$25% of the spam testing) using an indicator of spamminess over 0.50.
- The percentage of emails which obtained the indicator of spamminess over 0.60, got reduced to 0 in the case of the non-spam set and got reduced to 40.69 % for the spam testing set. The body of the emails seems to be a better indicator to determine if an email is spam or not. However, 40% is still a small subset of the spam testing set identified as spam. In other words, 60% of false negative results.
- From that point, as we increase the theshold indicator, the percentage of spam emails decrease in small rates. Those emails represent the obvious spam emails, which suggests that most of their words only found in our spam training set.

Table 3: Naive Bayesian Classifiers applied on the senders - SpamAssassin Dataset 2002

Set of Testing Emails	% SPAM ISender > 0.50	% SPAM ISender > 0.60	% SPAM ISender > 0.70	% SPAM ISender > 0.80	% SPAM ISender > 0.90
N.SPAM SET (1708)	0.29%	0.0%	0.0%	0.0%	0.0%
SPAM SET (1197)	95.91%	82.13%	82.13%	82.13%	82.13%

Some highlights considering only the sender of the emails to determine whether it is spam or not:

- We got a very small rate of false positive results (i.e. 0.29% of the non-spam testing set) and also this time a very low rate of false negative results (i.e. less than 5% of the spam testing) using an indicator of spamminess over 0.50.
- The percentage of emails which obtained the indicator of spamminess over 0.60, got reduced to 0 in the case of the non-spam set and slightly got reduced to 82.13

% for the spam testing set. This suggests that the sender of the email can be fairly used to determine if an email is spam or not. However, other techniques not covered in this paper (e.g. phising) may infringe this approach.
- From that point, as we increase the theshold indicator, we can see that the percentages of spam emails don't change, which means they were classified with a very high confidence of being spam (i.e. over 0.90).

Per the above results, we can not (and should not) merely determine if an email is spam or not based on one of its parts as a threshold of 0.5 doesn't give a good confidence. We found that we get better results if we assigned weights to each indicator and provide a unique indicator. For that purpose we defined the following weights:

$$I_w = (I_{sender} * 0.35 + I_{subject} * 0.15 + I_{body} * 0.5) \tag{17}$$

Based on our results, we assigned a considerable weight of 0.5 to the body of the email — as it is the main part involved. Additionally, we assigned a fair weight of 0.35 to the sender and a lower weight of 0.15 to the subject. Results are shown in table 4.

Table 4: Naive Bayesian Classifiers applied on the weighted parts of the email - SpamAssassin Dataset 2002

Set of Testing Emails	% SPAM Iw > 0.50	% SPAM Iw > 0.60	% SPAM Iw > 0.70	% SPAM Iw > 0.80	% SPAM Iw > 0.90
N.SPAM SET (1708)	1.23%	0.0%	0.0%	0.0%	0.0%
SPAM SET (1197)	96.91%	90.48%	53.38%	30.99%	21.97%

Some highlights considering the weighted parts of the email to determine whether it is spam or not:

- We still got a very small rate of false positive results (i.e. 1.23% of the non-spam testing set) and also a very low rate of false negative results (i.e. less than 4% of the spam testing) using an indicator of spamminess over 0.50.
- The percentage of emails obtaining the indicator of spamminess over 0.60, got reduced to 0 in the case of the non-spam set and slightly got reduced to 90.48 % for the spam testing set. This means that none of the non-spam emails and almost all the spam testing set was classified as spam (zero false positives and $\sim$10% of false negatives) using an threhold indicator of 0.6, which give us more confidence
- From that point, as we increase the indicator theshold, we can see how the percentages of spam emails reduce.

3.2 CSMINING emails dataset - 2010

Previous results were not a surprise, we basically used an approach proposed on 2003 with a contemporary dataset. The interesting part starts here, our second dataset was obtained from the CSMINING GROUP of 2010 [**CSMINING-2010**]. It contains 2947 non-spam emails and 1365 spam emails, from which 1461 non-spam and 687 spam emails were randomly chosen to train the tool. The rest of the emails were processed by the tool and the results are shown in tables 5, 6 and 7.

Again, the Naive Bayesian Classifiers method was idenpendently executed on each of the main parts of an email — the subject (table 5), the body (table 6) and the email sender (table 7). It is shown the percentage of spam emails considering different thresholds for the indicator of spamminess.

Table 5: Naive Bayesian Classifiers applied on the subjects - CSMINING Dataset 2010

Set of Testing Emails	% SPAM Isubject > 0.50	% SPAM Isubject > 0.60	% SPAM Isubject > 0.70	% SPAM Isubject > 0.80	% SPAM Isubject > 0.90
N. SPAM SET (1486)	0.53%	0.0%	0.0%	0.0%	0.0%
SPAM SET (678)	75.66%	39.68%	30.09%	25.07%	19.76%

Some highlights considering only the subject of the emails to determine whether an email is spam or not:

- We got almost the same results, a very small rate of false positive results (i.e. 0.53% of the non-spam testing set) and a fair false negative rate (i.e.$\sim$25% of the spam testing) using an indicator of spamminess over 0.50.
- As we saw with the previous dataset, the percentage of emails which obtained the indicator of spamminess over 0.60, dramatically got reduced on both subsets: to 0.0% of the non-spam and to 39.68% of the spam set. This means that most of the emails got an indicator of spamminess between 0.50 and 0.60. Consequently, the subject of the emails by itself can't be used to determine if an email is spam or not. Presumably because the words used can be found on both training sets.
- From that point, as we increase the indicator theshold, the percentage of spam emails decrease in small rates. Those emails represent the obvious spam emails. It is very likely that most of their words are only found in our spam training set.

Table 6: Naive Bayesian Classifiers applied on the bodies - CSMINING Dataset 2010

Set of Testing Emails	% SPAM IBody > 0.50	% SPAM IBody > 0.60	% SPAM IBody > 0.70	% SPAM IBody > 0.80	% SPAM IBody > 0.90
N.SPAM SET (1486)	0.61%	0.07%	0.0%	0.0%	0.0%
SPAM SET (678)	77.28%	43.36%	38.20%	32.15%	27.73%

Some highlights considering only the body of the emails to determine whether it is spam or not:

- Again, we got a very small rate of false positive results (i.e. 0.61% of the non-spam testing set) and a fair false negative rate (i.e. $\sim$23% of the spam testing) using an indicator of spamminess over 0.50. These results are very similar to those obtained with the 2002 dataset.
- The percentage of emails which obtained the indicator of spamminess over 0.60, got reduced to almost 0 in the case of the non-spam set and got reduced to 43.36 % for the spam testing set. The body of the emails seems to be a better indicator to determine if an email is spam or not. However, 43% is still a small subset of the spam testing set identified as spam. In other words, 57% of false negative results. Again, these results are very similar to those obtained with the 2002 dataset.
- From that point, as we increase the theshold indicator, the percentage of spam emails decrease in small rates. Those emails represent the obvious spam emails, which suggests that most of their words can only be found in our spam training set.

Table 7: Naive Bayesian Classifiers applied on the senders - CSMINING Dataset 2010

Set of Testing Emails	% SPAM ISender > 0.50	% SPAM ISender > 0.60	% SPAM ISender > 0.70	% SPAM ISender > 0.80	% SPAM ISender > 0.90
N.SPAM SET (1486)	5.92%	0.0%	0.0%	0.0%	0.0%
SPAM SET (678)	64.45%	64.45%	64.45%	64.45%	64.45%

Some highlights considering only the sender of the emails to determine whether it is spam or not:

- We got different rates here. A greater number of false positive results (i.e. 5.92% of the non-spam testing set vs 0.29%) and also a considerable number of false

negative results (i.e. ∼35% of the spam testing vs 5%) using an indicator of spamminess over 0.50.
- From that point, as we increase the theshold indicator, we can see how that the percentages of spam emails reduced to zero in the case of non-spam set and don't change in the case of spam emails set, which means they were classified with a very high confidence of being spam (i.e. over 0.90).

As we did with the previous dataset, we assigned weights to each indicator and provide a unique one. We defined the same weights described in equation 17. Results are shown in table 8.

Table 8: Naive Bayesian Classifiers applied on the weighted parts of the email - CSMINING Dataset 2010

Set of Testing Emails	% SPAM Iweighted > 0.50	% SPAM Iweighted > 0.60	% SPAM Iweighted > 0.70	% SPAM Iweighted > 0.80	% SPAM IWeighted > 0.90
N.SPAM SET (1486)	0.13%	0.0%	0.0%	0.0%	0.0%
SPAM SET (678)	96.61%	78.76%	52.06%	24.33%	16.96%

These results show that current spam emails are more difficult to detect. The rate of false negative got increased with recent emails (considering a threshold of 0.6). This is usually accepted by the users as they are more concerned in receiving all their important emails on their inbox. This is the reason why Naive Bayesian Classifiers are still a good option for identifying spam emails as only less than 22% of the spam emails were classified as non-spam (false negatives).

3.3 UNICAMP SMSs dataset - 2012

So far we have seen that current spam emails can still be identified pretty well with Naive Bayesian Classifiers method. The next question is if this same method can be used to identify current spam SMSs.

For that purpose, we took a dataset published by the Universidade Estadual de Campinas [**UNICAMP-2010**]. It contains 4826 non-spam SMSs and 747 spam SMSs, from which 2129 and 362 messages were randomly chosen to be used for training of the tool. The rest of the messages were processed by the tool and the results are shown in table 9.

It seems that it is harder to identify the spamminess of an SMS, presumably because of its lenght. We can see that there are much more false positives (around 4.5%) which obtained an indicator over 0.5. This may be a problem for the users

Table 9: Naive Bayesian Classifiers applied on SMSs - UNICAMP Dataset 2012

Set of Testing SMSs	% SPAM Isms > 0.50	% SPAM Isms > 0.60	% SPAM Isms > 0.70	% SPAM Isms > 0.80	% SPAM Isms > 0.90
N.SPAM SET (2697)	4.49%	0.11%	0.0%	0.0%	0.0%
SPAM SET (385)	95.58%	78.70%	74.02%	68.83%	62.34%

if that threshold is used. On the other hand, if a greater indicator is used, the users may miss only few legitimate sms, but they will still receive some spam messages (i.e. $\sim$20% - 25%).

4 Conclusions and Future Work

Naive Bayesian Classifiers is still a good method to identify spam emails and spam SMSs. Specifically because, current legitimate messages are still identified as non-spam which means the user does not miss any authentic message.

On the other hand, spam messages are now a little bit harder to detect (around %20 not detected). Mainly because spam emails are now sent with large amounts of legitimate text, which causes it to be likely classified as non-spam. Additionally, words are transformed. For example "Viagra" would be replaced with "Viaagra" or "V1agra" in the spam message. The recipient of the message can still read the changed words, but each of these words is met more rarely by the Bayesian filter, which obstructs its learning process.

Naive Classifiers can be enhanced with other learning techniques to reduce the number of missed spam messages.

References

1. Graham, Paul : A Plan for Spam (2002). http://paulgraham.com/.Cited16Nov2013
2. Robinson, Gary: A Statistical Approach to the Spam Problem. Belltown Medial (March 2003)
3. Robinson, Gary: Handling Redundancy in Email Token Probabilities (May 2004)
4. Peters, Tim: SpamBayes anti-spam (2004) http://spambayes.sourceforge.net/background.html.Cited16Nov2013
5. G. V. Cormack and T. R. Lynam: TREC 2005 Spam Track Overview (2005) http://plg.uwaterloo.ca/œgvcormac/trecspamtrack05.Cited16Nov2013
6. G. V. Cormack: Email Spam Filtering: A Systematic Review (2005)
7. MessageLabs Intelligence: 2010 Annual Security Report

8. MAAWG: Email Metrics Program: The Network Operator's Perspective. Report 14. http://www.maawg.org/sites/maawg/files/news/MAAWG_2010_Q3Q4_Metrics_Report_14.pdf.Cited16Nov2013
9. Nucleus Research Inc: Spam: The Silent ROI Killer (July 2003) http://www.nucleusresearch.com/research/d59.pdf.Cited16Nov2013
10. The Apache SpamAssassin Project: Spam dataset (2002) http://spamassassin.apache.org/publiccorpus/.Cited16Nov2013
11. CSMINING GROUP: Spam dataset (2010) http://csmining.org/index.php/spam-email-datasets-.html.Cited16Nov2013
12. UNICAMP: SMS Spam Collection (2012) http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/.Cited16Nov2013

Comparación de Dos Métodos de Filtrado de Spam Basados en Bayes

Carlos Alberto García Márquez

1 Introducción

En la actualidad, el correo electrónico se ha convertido en un medio de comunicación relevante para la vida diaria de las personas. Tanto en el ámbito personal como dentro de ambientes de negocio y trabajo en general.

La utilización del correo electrónico ha venido a ser una práctica cada vez más extendida entre las personas y a la que se recurre cada vez más para actividades de comunicación cotidiana.

Además de la caracteríastica importancia per se en la sociedad actual con que cuenta, el correo electrónico también se puede convertir en una amenaza latente para la privacidad de las personas as como para la confidencialidad y la integridad de la información, ya que la utilización de correo electrónico masivo más conocido como correo basura o spam (en adelante spam), además de la afectación implícita que por sí misma lleva por costos de tiempo, ha venido a ser la herramienta predilecta para la práctica de actividades fraudulentas, tales como el Phishing.

Para evitar que el correo spam llegue a las bandejas de entrada de correo electrónico existen diversas técnicas de filtrado de correo, como las basadas en reglas, en aprendizaje, entre otras (como la propuesta por Zhou en [**Zhou07**]).

En la actualidad, los métodos basados en el aprendizaje han jugado un papel destacado en la tarea del filtrado de spam puesto que han permitido llevar a cabo el filtrado de correo de una manera dinámica a la vez que los parámetros utilizados para la detección se pueden ir ajustando a las preferencias del usuario.

Dentro del conjunto de técnicas utilizadas para llevar a cabo el filtrado de correo electrónico, las que se basan en el Teorema de Bayes han sido objeto de estudio y mltiples trabajos de investigación con resultados muy interesantes y un buen rendimiento en la detección de spam.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

En el presente trabajo se realiza una comparación entre dos algoritmos para el filtrado de mensajes basados en el teorema de Bayes. El primer método de filtrado de correo electrónico está basado en el propuesto por Androutsopoulos en [**Androutsopoulos1**] el cuál a su vez retoma parte del trabajo presentado por Sahami en [**Sahami1998**], mientras que el segundo se basa en el propuesto por Graham en [**Graham02**].

Ambos algoritmos Bayesianos son orientados al aprendizaje y se apoyan en técnicas de valoración del impacto negativo causado por la clasificación errónea de mensajes (correo legítimo como spam o spam no filtrado).

2 Objetivos

2.1 Objetivo General

El presente trabajo tiene como objetivo determinar las diferencias de rendimiento para el filtrado de correo spam entre el algoritmo propuestos por Androutsopoulos en [**Androutsopoulos1**] y el algoritmo propuesto por Graham en [**Graham02**], con el fin de determinar en qué circunstancias se puede recomendar su uso en base a la efectividad del filtrado lograda con relación a la precisión obtenida.

2.2 Objetivos Específicos

Para ambos algoritmos se pretende:

1. Determinar la efectividad del filtrado (relacionado con la tasa de falsos negativos).
2. Determinar la precisión del filtrado (relacionado con la tasa de falsos positivos).
3. Realizar una comparación de rendimiento entre ambos algoritmos.
4. Determinar las circunstancias en que se pudiera recomendar su uso.

3 Teorema de Bayes

El teorema de Bayes define la realización de cálculos probabilísticos tomando en consideración la evaluación de información adicional recopilada posteriormente, es decir, se orienta a la evaluación de eventos secuenciales en que la información nueva que se obtiene de eventos posteriores es usada para confirmar o revisar la probabilidad del evento anterior.

La fórmula básica para llevar a cabo el cálculo de la probabilidad de un evento A dada la posterior ocurrencia de un evento B de acuerdo a este teorema es la siguiente:

$$P(A|B) = \frac{P(A) * P(B|A)}{[P(A) * P(B|A)] + [P(\overline{A}) * P(B|\overline{A})]} \quad (1)$$

Esta fórmula nos permite calcular la probabilidad de que el evento A haya sucedido dado un evento B con base en la probabilidad de que B se haya dado a partir de A, y de A prima (la probabilidad de no A).

3.1 Ejemplo de Aplicación del Teorema de Bayes

Se puede explicar el teorema de Bayes mediante el siguiente ejemplo (tomado de [**Triola10**]):

Si de una población de personas, el 51% son hombres y el 49 por ciento son mujeres, además, el 9.5% de los hombres son fumadores mientras que el 1.7% de las mujeres son fumadoras, la probabilidad de que al entrevistar a un individuo que fuma, éste sea hombre se calcula por el Teorema de Bayes de la siguiente manera:

A - Población de hombres.
$\overline{A}$ - Poblacion de mujeres.
B - Fumador(a).
$P(A) = 0.51$
$P(B|A) = 0.095$
$P(\overline{A}) = 0.49$
$P(B|\overline{A}) = 0.017$

$$P(A|B) = \frac{0.51 * 0.095}{[(0.51 * 0.095) + (0.49 * 0.017)]} \quad (2)$$

$$P(A|B) = \frac{0.04845}{0.04845 + 0.00833} \quad (3)$$

$$P(A|B) = \frac{0.04845}{0.05678} \quad (4)$$

$$P(A|B) = 0.85329341317365269461077844311377 \quad (5)$$

Redondeando, la probabilidad de que una persona entrevistada que se sabe que es fumadora sea un hombre en base a los datos que se tiene es de 0.8532 ó $P(A|B) = 0.8532$.

4 Dos Herramientas para el Filtrado de Correo Spam Basadas en el Teorema de Bayes

4.1 Filtro Basado en el Algoritmo propuesto por Androutsopoulos

Para la realización de este filtro de correo se establece que cada mensaje se representa por un vector $\mathbf{x}\{x_1, x_2, x_3, \ldots, x_n\}$, donde $x_1, \ldots, x_n$ son valores para atributos $X_1, \ldots, X_n$. De acuerdo a Androutsopoulos [**Androutsopoulos1**] y Sahami [**Sahami1998**], se asignan valores $X_i = 1$ si algunas caractersticas de X_i se encuentran en el mensaje, de manera contraria $X_i = 0$.

Para el presente trabajo, retomando lo expuesto por Androutsopoulos en [**Androutsopoulos1**], los atributos tomados corresponden a palabras que son buscadas en el contenido del correo, por ejemplo adult.

Para realizar la seleccin entre todos los atributos posibles, se sigui la misma lnea propuesta por Androutsopoulos [**Androutsopoulos1**] y Sahami [**Sahami1998**], procedindose a calcular la informacin mutua (MI) de cada atributo candidato X con una denotación de categora C:

$$MI(X;C) = \sum_{x \in 1,0, c \in spam, legitimo} P(X = x, C = c) * log \frac{P(X = x, C = c)}{P(X = x) * P(C = c)} \quad (6)$$

Donde los atributos con un mayor valor para MI son seleccionados. Las probabilidades son calculadas.

Siguiendo a Androutsopoulos [**Androutsopoulos1**], mediante la utilización del teorema de Bayes, dado el vector $\mathbf{x}\{x_1, \ldots, x_n\}$, de un documento d y suponiendo que $X_1, \ldots, X_n$ son condicionalmente independientes dada una categoría C, la probabilidad de que dicho documento d pertenezca a una categoría c se puede obtener mediante:

$$P(C = c|\mathbf{X} = \mathbf{x}) = \frac{P(C = c) * \prod_{i=1}^{n} P(X - i = x_i|C = c)}{\sum_{k \in \{legitimo, spam\}} P(C = k) * \prod_{i=1}^{n} P(X_i = x_i|C = k)} \quad (7)$$

Siguiendo a Androutsopoulos [**Androutsopoulos1**], para nuestra herramienta de filtrado de correo se hace la suposición simplificada de que $X_1, \ldots, X_n$ son condicionalmente independientes dada una categoría C.

En cuanto a la evaluación del impacto de un error en el filtrado de mensajes, es necesario tomar en cuenta que no representa el mismo costo para el usuario el hecho de clasificar un correo legítimo como spam que el error contrario (clasificar un correo spam como legítimo) de acuerdo a lo sealado por Androutsopoulos [**Androutsopoulos1**].

El hecho de clasificar un correo legítimo como spam representa un costo mayor para el usuario que el hecho inverso, por lo que es necesario cuantificar y cualificar de alguna manera este costo. Para llevar a cabo la cuantificación del costo, fue tomada la fórmula utilizada por Androutsopoulos [**Androutsopoulos1**], con-

siderando $L \rightarrow S$ (el error de clasificar correo legítimo como spam) y $S \rightarrow L$(el error de clasificar correo spam como legítimo).

Siguiendo con Androutsopoulos [**Androutsopoulos1**], se tom $L \rightarrow S$ como λ veces más costoso que $S \rightarrow L$, por lo que se clasificará un mensaje como spam sólo si:

$$\frac{P(C = spam|\mathbf{X} = \mathbf{x})}{P(C = legitimo|\mathbf{X} = \mathbf{x})} > \lambda \tag{8}$$

En este trabajo, dados los posibles valores, $P(C = spam|\mathbf{X} = \mathbf{x}) = 1 - P(C = legitimo|\mathbf{X} = \mathbf{x})$, se puede reformular el criterio de clasificación como spam obteniendo:

$$P(C = spam|\mathbf{X} = \mathbf{x}) > t, \text{ donde } t = \frac{\lambda}{1+\lambda}, \lambda = \frac{t}{1-t} \tag{9}$$

Siguiendo a Sahami [**Sahami1998**] y Androutsopoulos [**Androutsopoulos1**], se utilizaron los límites $t = 0.999$ ($\lambda = 999$), lo que indica que el bloqueo de un correo legítimo el cuál es borrado sin un tratamiento posterior es tan grave como dejar pasar 999 mensajes spam, $t = 0.9$ ($\lambda = 9$) en el caso en que al bloquear y borrar un correo legítimo se envía una respuesta al remitente solicitando el reenvío del correo a una dirección de correo alterna libre de amenaza de spam (dirección de correo no publicada) con el empleo de un reto adicional (tal como indicar el nombre de la capital de Inglaterra) es tan grave como dejar pasar 9 correos spam con la suposición de que para el remitente de correo legítimo implica un costo el reenvío de su correo y $t = 0.5$ ($\lambda = 1$) cuando al igual que en el caso anterior se solicita el reenvío del correo lo que no representa costo para el remitente [**Androutsopoulos1**].

Más adelante se presentarán los resultados obtenidos con la aplicación de este método.

4.2 Filtro Basado en el Algoritmo propuesto por Graham

Mediante la utilización del algoritmo propuesto por Graham [**Graham02**] se realiza una clasificación de palabras por probabilidades, pero se otorga un peso de 2 a 1 a la probabilidad calculada para el uso de palabras en correo legítimo contra el uso en correo spam. El valor de 2 es un heurístico que Graham logró determinar por ensayo y error de acuerdo a lo mencionado por él en [**Graham02**].

El hecho de utilizar un multiplicador de 2 para calcular el porcentaje de ocurrencia en correo legítimo para una palabra le permite evitar falsos positivos [**Graham02**].

Para calcular la probablidad de ocurrencia para una palabra dentro de un correo spam Graham propone el siguiente algoritmo (tomado de [**Graham02**]):

$(let((g(*2(or(gethashwordgood)0)))$
$(b(or(gethashwordbad)0)))$
$(unless(< (+gb)5)$
$(max0.01$

$(min0.99(float(/(min1(/bnbad))$
$(+(min1(/gngood))$
$(min1(/bnbad)))))))))$

Donde *good* y *baad* son tablas hash en que se almacenan las palabras encontradas en correo legítimo y spam respectivamente, mientras que *nbad* y *ngood* son las cantidades de correo spam y y legítimo respectivamente y word es el identificador referido a la palabra que se evalúa.

Para mayor claridad se puede ver la implementación del algoritmo en Java de la siguiente manera:

```
1  public void setLegProb(double total) {
2        if (total > 0){
3              porcentLegit = 2*contLeg / (float) total;
4        }
5  }
6
7  public void estabSpamProb() {
8        if (porcentLegit + porcentSpam > 0) {
9              probSpam = porcentSpam / (porcentSpam +
                  porcentLegit);
10       }
11       if (probSpam < 0.01f){
12             probSpam = 0.01f;
13       }
14       else if (probSpam > 0.99f){
15             probSpam = 0.99f;
16       }
17 }
```

Como puede observarse, la determinación de la probabilidad de ocurrencia de una palabra en correo legítimo es multiplicada por 2 tal como se había mencionado.

Para realizar el cálculo de la probabilidad combinada de las palabras presentes en el correo que se está analizando, Graham propone el siguiente algoritmo [**Graham02**]:

$(let((prod(apply\sharp' * probs)))$
$(/prod(+prod(apply\sharp' * (mapcar\sharp'(lambda(x)$
$(-1x))$
$probs)))))$

A continuación se muestra la implementación en lenguaje Java de este algoritmo para mayor claridad:

```
1  double productPos = 1.0D;
2  double productNeg = 1.0D;
3
4  for (int i = 0; i < sospechosas.size(); i++) {
5        PalabraGraham_2 palabra = (PalabraGraham_2)
            sospechosas.get(i);
```

```
6         productPos *= palabra.getProbSpam();
7         productNeg *= (1.0D - palabra.getProbSpam());
8
9   }
10  probSpam = productPos / (productPos + productNeg);
11  return (probSpam > this.limite);
```

Graham propone utilizar las 15 palabras con los valores mayores de probabilidad de spam para realizar la clasificación del correo para determinar la pertenencia a la categoría de spam [**Graham02**]. Se hizo un ajuste a esta parte del algoritmo con el fin de utilizar diferentes cantidades de atributos (palabras) para hacer la clasificación.

En su algoritmo original Graham propone utilizar un lmite de $t = 0.9$ [**Graham02**] ($\lambda = 9$) por lo que se hizo una adaptación al algoritmo con el fin de poder utilizarlo con valores para $t = 0.999$ ($\lambda = 999$) y $t = 0.5$ ($\lambda = 1$) para utilizar los valores para el límite tal como se utilizaron para la implementación del algoritmo de Androutsopoulos.

El algoritmo de graham selecciona aquellas palabras con una probabilidad de pertenencia a correo spam mayor a 0.5 como palabras candidatas a ser spam y de las palabras seleccionadas se filtran aquellas que poseén el mayor valor de probabilidad a ser spam.

Más adelante se presentarán los resultados obtenidos con la aplicación de este método.

5 Los Datos de Muestra Utilizados

Resulta difícil la utilización del correo contenido en la bandeja de entrada de alguna persona para realizar las pruebas de detección de correo electrónico spam debido a las implicaciones de privacidad que existen de por medio, por lo que usualmente se utilizan corpus de correo previamente recopilado.

Anteriormente Androutsopoulos en [**Androutsopoulos1**] había utilizado un corpus de correo que se identifica como Ling-Spam el cuál es accesible de manera gratuita. Este es el corpus que se utilizó para la realización del presente trabajo.

Existen actualmente varias versiones actualizadas de dicho corpus las cuales se consiguieron de la dirección electrónica http://csmining.org/index.php/ling-spam-datasets.html.

El corpus utilizado es el que se encuentra accesible en la primera opción de descarga del sitio (lingspam_public.tar.tar).

Este corpus contiene en su interior cuatro carpetas que se refieren a las cuatro opciones de filtrado con que trabajó Androutsopoulos en [**Androutsopoulos1**], las cuales incluían el trabajo con correos cuyas palabras fueron convertidas a su base y los módulos de utilización de lemas más lista de bloqueo, utilización de lemas sin lista de bloqueos y lista de bloqueos [**Androutsopoulos1**]. Se utlizaron los correos

contenidos en las cuatro carpetas para correr las pruebas para las dos opciones de filtrado.

Dentro de cada una de las cuatro carpetas se encuentran diez carpetas, todas con ejemplos de correo electrónico de los que aquellos que son correo spam contienen en el nombre de archivo el prefijo ”spmsg”, mientras que el correo legítimo contienen en el nombre de archivo el descriptivo ”msg” únicamente como parte del nombre.

En total, el corpus cuenta con 11,573 archivos, de los que 1,925 (al rededor del 16.6 por ciento) son correo spam.

De las diez carpetas por cada uno de los conjuntos se utilizaron las primeras 9 para realizar el entrenamiento de los algoritmos y la carpeta restante fué la utilizada para realizar la prueba final del funcionamiento.

6 Resultados

Se realizaron las mismas pruebas para ambos algoritmos, obteniéndose resultados satisfactorios para ambos filtros con sus respectivas diferencias relacionadas con las diferencias en la estructura misma de éstos.

De acuerdo a Androutsopoulos [**Androutsopoulos1**] Si se considera $n_L \rightarrow_S$ y $n_S \rightarrow_L$ como el números de errores de $L \rightarrow S$ y $S \rightarrow L$ respectivamente y consideramos a $n_L \rightarrow_L$ y $n_S \rightarrow_S$ como los totales de correo tratado correctamente, se puede en base a esto calcular los valores de Spam Recall (SR) y Spam Precission (SP) como sigue:

$$SR = \frac{n_S \rightarrow_S}{n_S \rightarrow_S + n_S \rightarrow_L}; SP = \frac{n_S \rightarrow_S}{n_S \rightarrow_S + n_L \rightarrow_S} \tag{10}$$

En las figuras 1 y 2 se muestran los resultados graficados para Spam Recall contran Spam precission de ambos algoritmos utilizando un valor de $t = 0.5$, mientras que las figuras 3 y 4 muestran la misma grfica para un valor de $t = 0.9$ y las figuras 5 y 6 muestran las gráficas con los resultados para el valor de $t = 0.999$.

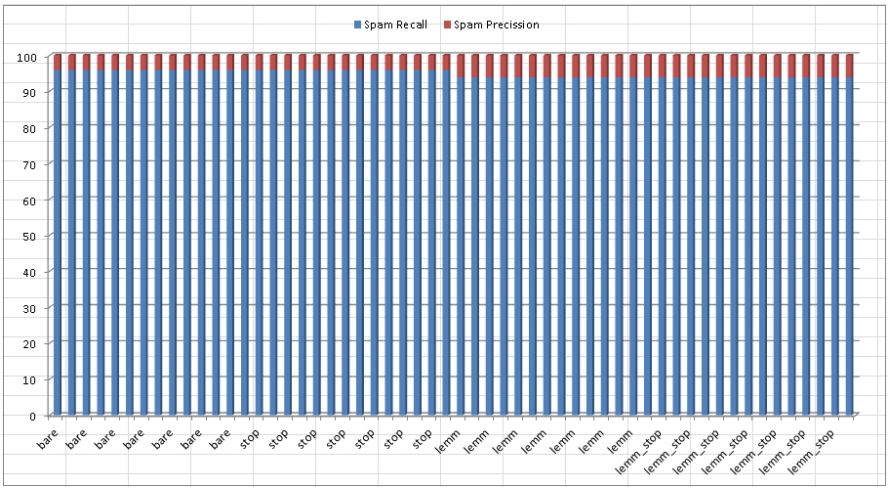


Fig. 1: Androutsopoulos - Spam Recall en relación con Spam Precission usando $t = 0.5$

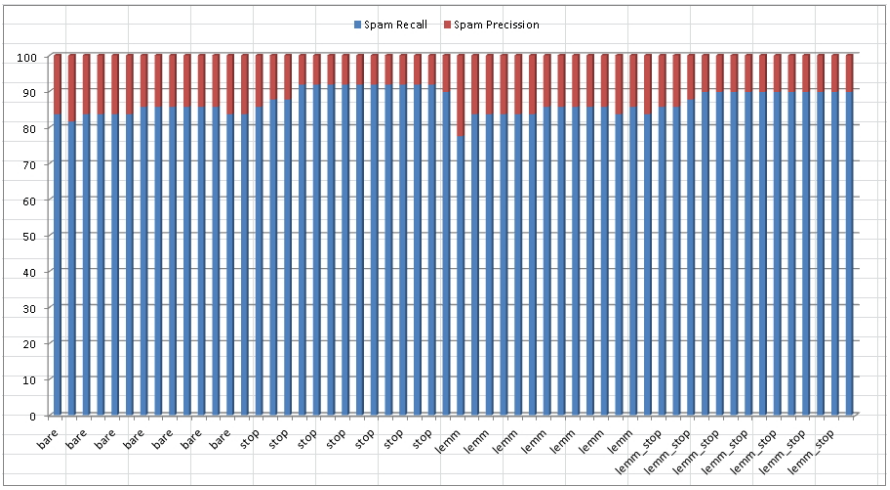


Fig. 2: Graham - Spam Recall en relación con Spam Precission usando $t = 0.5$

Las gráficas anteriores registran los resultados para diferentes ejecuciones tomando para la clasificación del correo desde 50 hasta 700 atributos con incrementos de 50 atributos entre cada ejecución.

En general, los resultados obtenidos mediante el uso del filtro que implementa el algoritmo de Androutsopoulos registra un rendimiento casi constante, con algunas leves diferencias, mientras que la versión del algoritmo de Graham utilizado registra

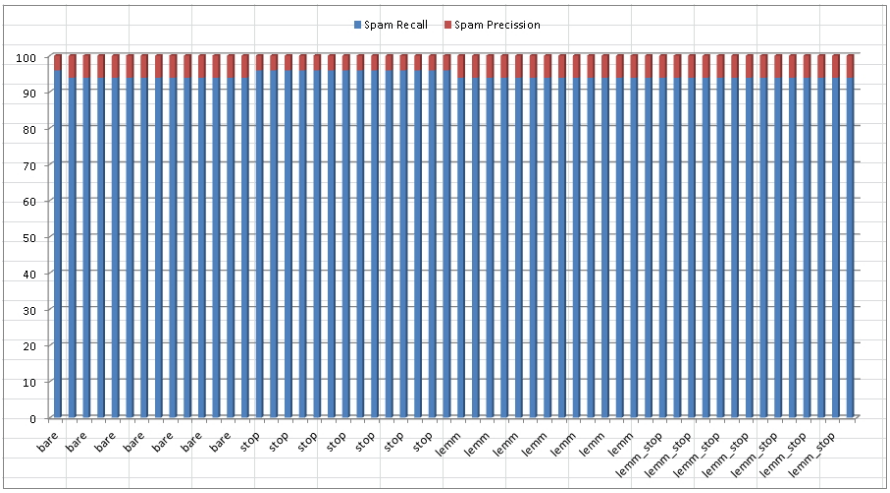


Fig. 3: Androutsopoulos - Spam Recall en relación con Spam Precission usando $t = 0.9$

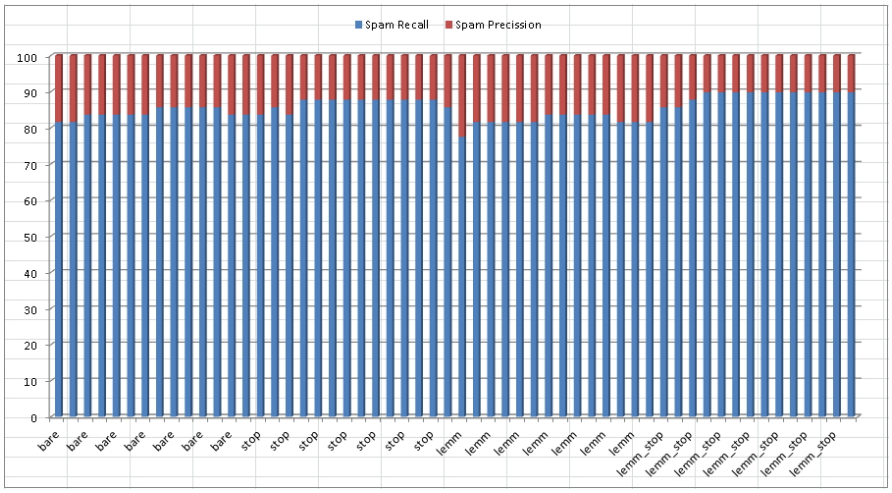


Fig. 4: Graham - Spam Recall en relación con Spam Precission usando $t = 0.9$

variaciones pronunciadas en el rendimiento además de mostrar valores de Spam Recall más bajos.

Dos indicadores comúnmente utilizados para la evaluación de tareas de clasificación son la precisión (Acc o accuracy en inglés) y el error (Err). Para obtener estos indicadores, se realizó el mismo cálculo realizado por Androutsopoulos [**Androutsopoulos1**] para ambos algoritmos:

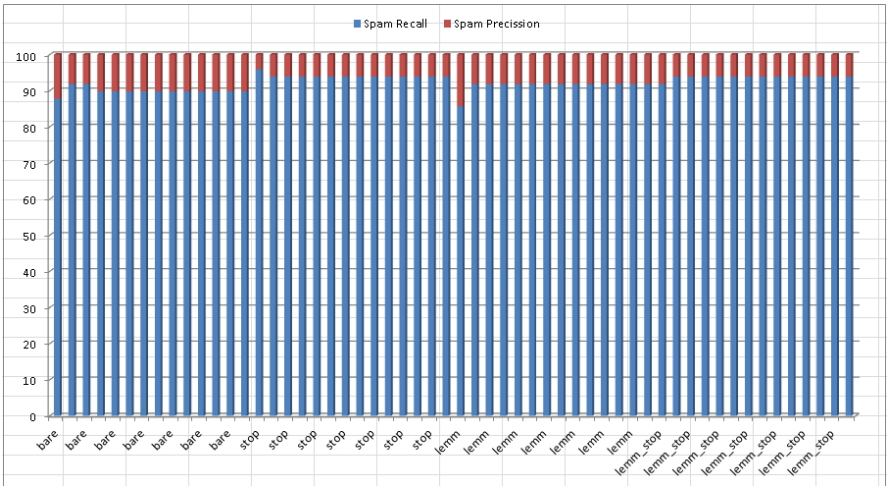


Fig. 5: Androutsopoulos - Spam Recall en relación con Spam Precission usando $t = 0.999$

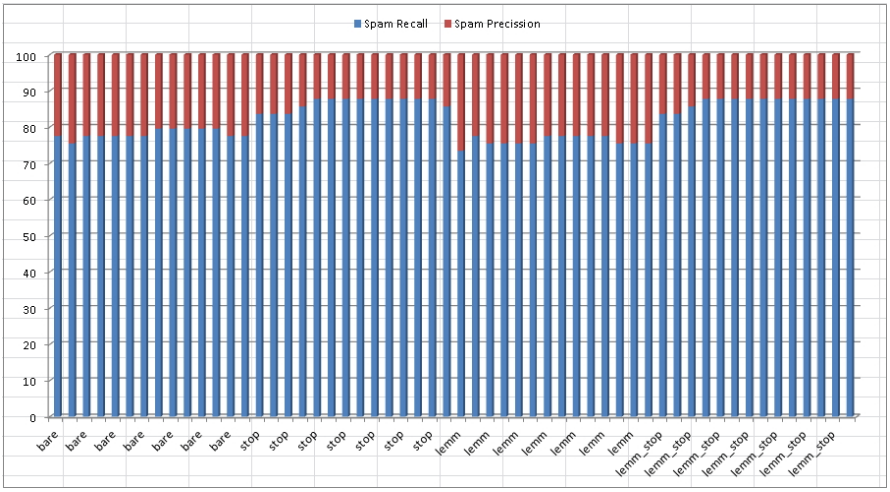


Fig. 6: Graham - Spam Recall en relación con Spam Precission usando $t = 0.999$

$$Acc = \frac{n_L \rightarrow_L + n_S \rightarrow_S}{N_L + N_S}; Err = \frac{n_L \rightarrow_S + n_S \rightarrow_L}{N_L + N_S} \tag{11}$$

Donde N_L y N_S son el número de correo legítimo y correo spam a ser clasificado [**Androutsopoulos1**]

Estos indicadores asignan el mismo peso a los dos tipos de error ($N_L \rightarrow_S, N_S \rightarrow_L$), pero nosotros asumimos que $N_L \rightarrow_S$ es λ veces más severo que $N_S \rightarrow_L$, por lo que aplicamos la fórmula propuesta por Androutsopoulos [**Androutsopoulos1**] para

obtener la precisión y error ponderados ($WAcc$ y $WErr$ respectivamente) donde $WErr = 1 - WAcc$ [**Androutsopoulos1**]:

$$WAcc = \frac{\lambda * n_L \rightarrow_L + n_S \rightarrow_S}{\lambda * N_L + N_S}; WErr = \frac{\lambda * n_L \rightarrow_S + n_S \rightarrow_L}{\lambda * N_L + N_S} \tag{12}$$

Es preciso comparar la precisión y el error obtenidos contra una línea base ($WAcc^b$ y $WErr^b$ respectivamente) para obtener una comparación válida que ayude a evitar interpretar incorrectamente los valores obtenidos para estos indicadores. Al comparar el error ponderado contra el error ponderado base podemos obtener un nuevo indicador que es la proporción del costo total (TCR o Total Cost Ratio en inglés) [**Androutsopoulos1**]. Se procede primero a calcular $WAcc^b$ y $WErr^b$ para posteriormente obtener TCR[**Androutsopoulos1**]:

$$WAcc^b = \frac{\lambda * N_L}{\lambda * N_L + N_S}; WErr^b = \frac{N_S}{\lambda * N_L + N_S} \tag{13}$$

Se obtiene TCR con la siguiente fórmula:

$$TCR = \frac{WErr^b}{WErr} = \frac{N_S}{\lambda * n_L \rightarrow_S + n_S \rightarrow_L} \tag{14}$$

De acuerdo con Androutsopoulos, a mayor valor de TCR corresponde un mejor desempeo en la clasificación de correo.

Las tablas 1 y 2 muestran los valores obtenidos para los indicadores para ambos algoritmos

Base de Datos	Limite	Atributos Usados	Spam Recall	Spam Precission	Exactitud Ponderada %	Exactitud Ponderada Base %	TCR
bare	0.5	50	95.91	95.91	98.62	83.16	12.25
stop	0.5	50	95.91	78.33	94.84	83.16	3.26
lemm	0.5	100	93.87	100	98.96	83.16	16.33
lemm_stop	0.5	100	93.87	100	98.96	83.16	16.33
bare	0.9	200	93.87	100	99.86	97.79	16.33
stop	0.9	200	95.91	100	99.91	97.79	24.5
lemm	0.9	100	93.87	100	99.86	97.79	16.33
lemm_stop	0.9	100	93.87	100	99.86	97.79	16.33
bare	0.999	200	89.79	100	99.99	99.97	9.8
stop	0.999	200	93.87	100	99.99	99.97	16.33
lemm	0.999	300	91.83	100	99.99	99.97	12.25
lemm_stop	0.999	300	93.87	100	99.99	99.97	16.33

Fig. 7: Tabla 1: Resultados utilizando el algoritmo de Androutsopoulos

Base de Datos	Limite	Atributos Usados	Spam Recall %	Spam Precission %	Exactitud Ponderada %	Exactitud Ponderada Base %	TCR
bare	0.5	50	83.67	100	97.25	83.16	6.12
stop	0.5	50	85.71	100	97.59	83.16	7
lemm	0.5	100	83.67	100	97.25	83.16	6.12
lemm_stop	0.5	100	85.71	100	97.59	83.16	7
bare	0.9	200	83.67	100	99.64	97.79	6.12
stop	0.9	200	87.75	100	99.73	97.79	8.16
lemm	0.9	100	81.63	100	99.59	97.79	5.44
lemm_stop	0.9	100	85.71	100	99.68	97.79	7
bare	0.999	200	77.55	100	99.99	99.97	4.45
stop	0.999	200	85.71	100	99.99	99.97	7
lemm	0.999	300	75.51	100	99.99	99.97	4.08
lemm_stop	0.999	300	87.75	100	99.99	99.97	8.16

Fig. 8: Tabla 2: Resultados utilizando el algoritmo de Graham

.

Las figuras 9 a 14 muestran los valores de la proporción del costo total en comparación con la cantidad de atributos evaluados para realizar la clasificación. Los valores mostrados son los correspondientes a ambos algoritmos.

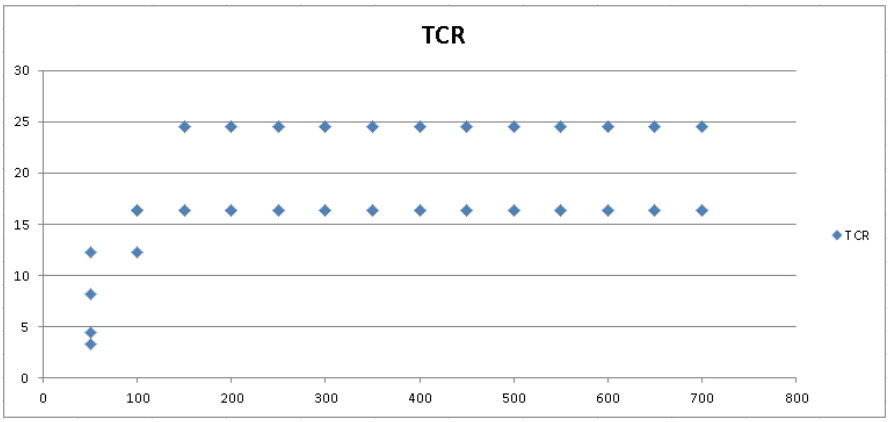


Fig. 9: Androutsopoulos - Total Cost Ratio por cantidad de atributos usando $t = 0.5$

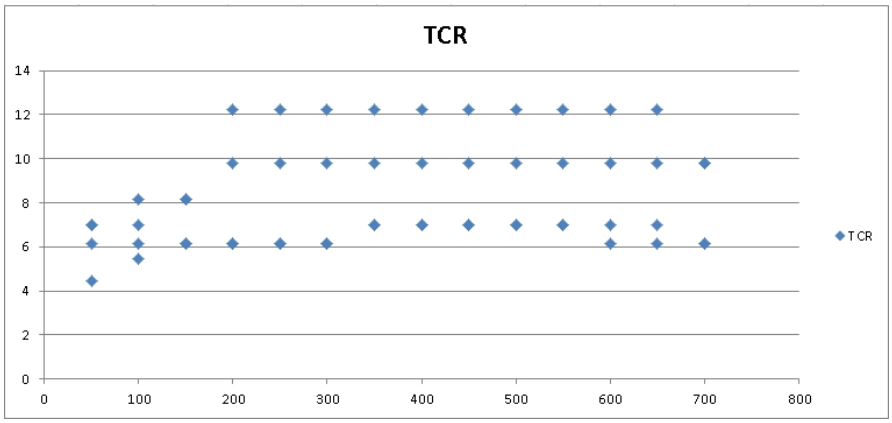


Fig. 10: Graham - Total Cost Ratio por cantidad de atributos usando $t = 0.5$

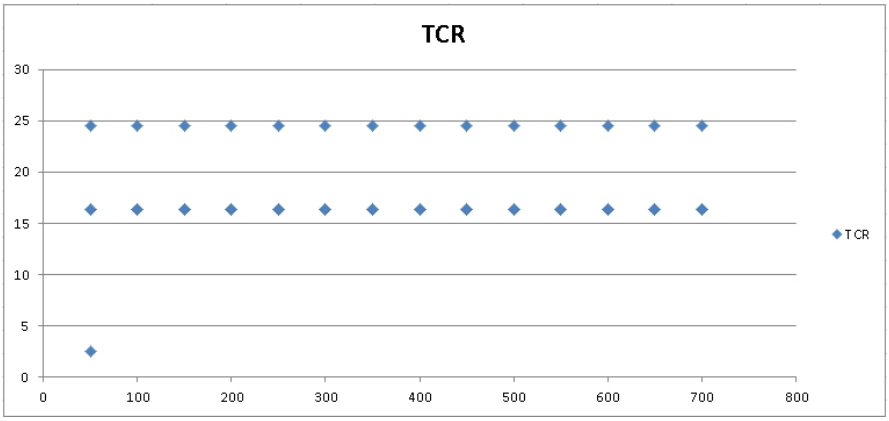


Fig. 11: Androutsopoulos - Total Cost Ratio por cantidad de atributos usando $t = 0.9$

Finalmente las figuras 15 y 16 muestran el sumarizado del TCR que incluye el TCR para la ejecución del algoritmo correspondiente con los diferentes valores de t.

7 Conclusiones

En base a lo expuesto es posible concluir lo siguiente:

De las versiones de los algoritmos de Androutsopoulos y Graham utilizados para el presente trabajo, el que obtuvo un mejor rendimiento en cuanto a spam recall fue el algoritmo de Androutsopoulos.

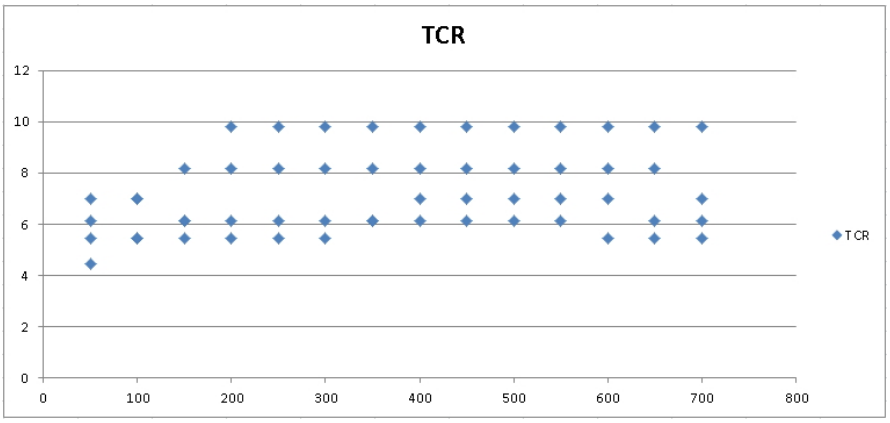


Fig. 12: Graham - Total Cost Ratio por cantidad de atributos usando $t = 0.9$

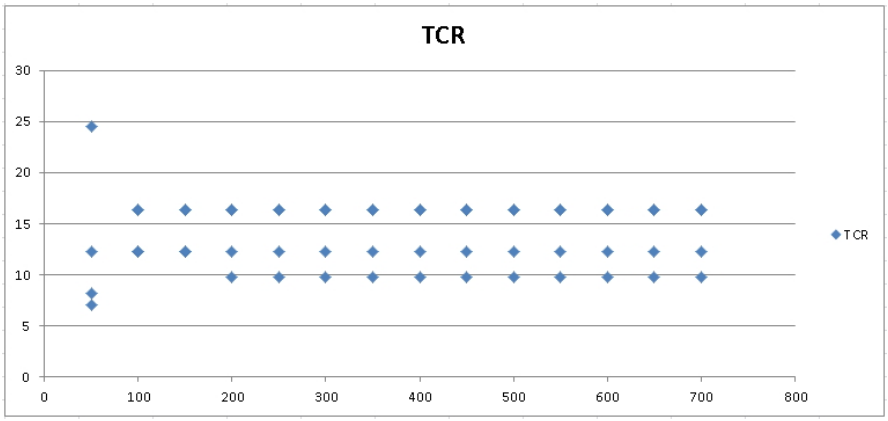


Fig. 13: Androutsopoulos - Total Cost Ratio por cantidad de atributos usando $t = 0.999$

Ambos algoritmos obtuvieron un buen rendimiento en cuanto a spam precission, aunque el algoritmo de Graham obtuvo el mejor rendimiento en general para este indicador obteniéndose siempre el 100 por ciento con este algoritmo, mientras que con el algoritmo de Androutsopoulos no se obtuvo el 100 por ciento de spam precission en todos los casos.

En cuanto a la proporción de costo total, el mejor desempeo se obtuvo con la versión del algoritmo de Androutsopoulos utilizada.

Pudiera decirse que el filtro basado en el algoritmo de Androutsopoulos es el recomendado cuando se requiere mayor rendimiento en TCR y Spam Recall cuando el correo filtrado recibe un tratamiento adicional tal como la solicitud de reenvío a una dirección de correo alternativa más un reto (cuando el valor del límite t es igual a

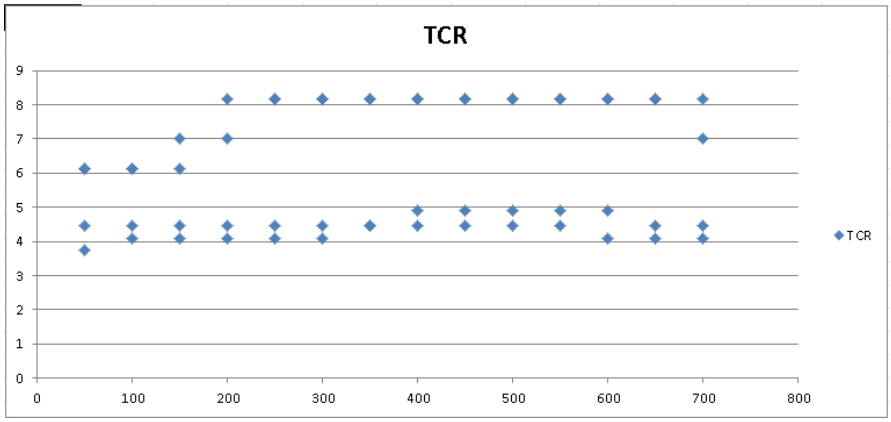


Fig. 14: Graham - Total Cost Ratio por cantidad de atributos usando $t = 0.999$

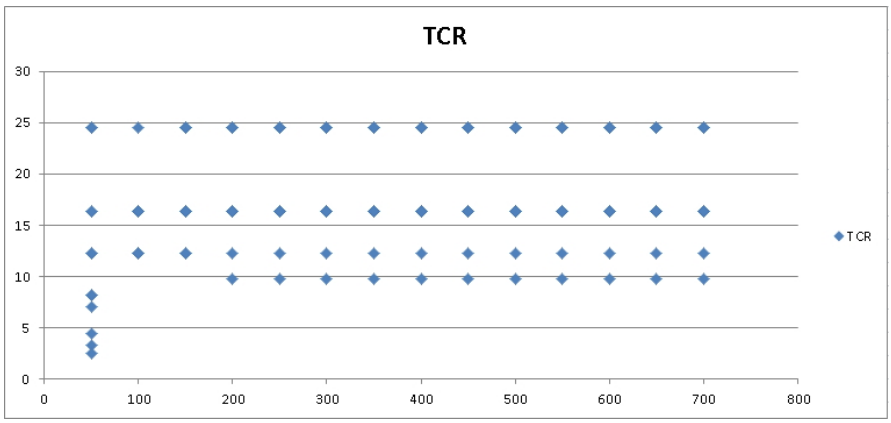


Fig. 15: Androutsopoulos - Sumarizado del Total Cost Ratio por cantidad de atributos

0.9 ó 0.5) debido a que los valores de Spam Precission no siempre aseguran el 100% de efectividad, mientras que en base a lo observado, si se pretende lograr niveles de Spam Precission altos, pudiera considerarse la alternativa de utilizar la herramienta basada en el algoritmo de Graham, el cuál en todas las pruebas realizadas obtuvo el 100% de Spam Precission.

Es recomendable realizar una comparación con algún otro método de clasificación de mensajes (como el expuesto por Zhou en [**Zhou07**]) con el fin de tener una referencia adicional de rendimiento independiente al teorema de Bayes.

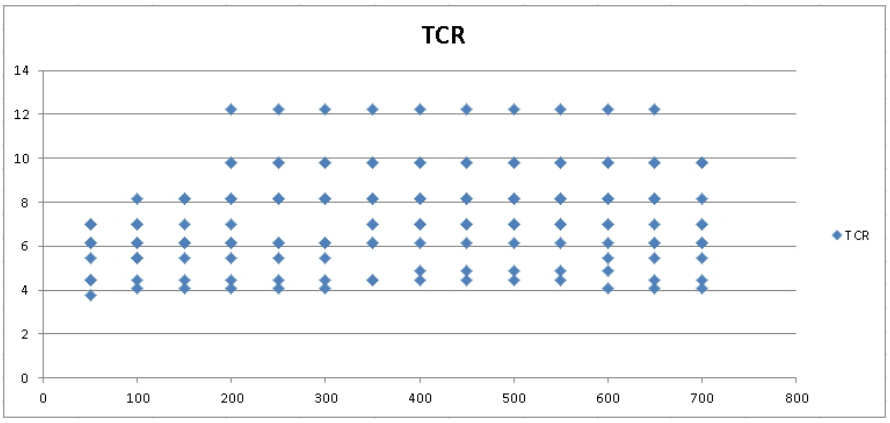


Fig. 16: Graham - Sumarizado del Total Cost Ratio por cantidad de atributos

References

1. Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. "A Bayesian Approach to Filtering Junk EMail". *In Learning for Text Categorization Papers from the AAAI Workshop, pp. 5562, Madison Wisconsin. AAAI Technical Report WS-98-05*, 1998.
2. Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras and Constantine D. Spyropoulos. "An Evaluation of Naive Bayesian Anti-Spam Filtering". *Proceedings of the workshop on Machine Learning in the New Information Age, G. Potamias, V. Moustakis and M. van Someren (eds.), 11 th European Conference on Machine Learning, Barcelona, Spain, pp. 9-17*, 2000.
3. Paul Graham, "A Plan for Spam". http://www.paulgraham.com/spam. 2002.
4. Mario F. Triola. "Bayes' Theorem". *Pearson Education*, 2010.
5. Ion Androutsopoulos, John Koutsias, Konstantinos V. Cbandrinos and Constantine D. Spyropoulos. "An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages Software and Knowledge" *Engineering Laboratory Institute of Informatics and Telecommunications National Centre for Scientific Research "Demokritos" 153 l0 Ag. Paraskevi, Athens, Greece*, 2000.
6. Xiao Zhou, "A Self-learning Spam Detecting System Model based on Memory Rule". *Jianmei Shuai School of Information Science and Technology University of Science and Technology of China, Hefei, China*, 2007.

Spam filtering based on Naive Bayesian method

Francisco Estrada Perez

Abstract The web mail service is as old as the internet itself. Since its creation some junk e-mail has been sent into multiple targets even when they have not requested it. This kind of e-mail also known as spam; is one of the most important tools that marketing companies use these days to inform and make a notice of their brand. There exist ways to filter and delete these kind of e-mails for ”normal” customers that have no interests in receiving this information. These are called Spam filters that identify junk e-mail, usually by identifying some key words and classifying the message as a non legit e-mail or spam. In this paper we discuss some of the existing filters used and specially the Naive Bayesian method and how it works. We tested it against a e-mail pool to observe its efficiency. We will discuss the results and conclude how could we improve the implementation and the importance of understanding how to fight spam.

1 Introduction

We use e-mail everyday as form of communication. Since the invention of the internet web mail has been used and is growing everyday to allow people to send and receive information quick, easy and in a cheap way. As a everyday user we employ the e-mail tool everyday and we even have multiple accounts related to different servers. We use an account for our job, one for personal reasons and some people even have a different one for school. All of these accounts help us communicate important topics from homework, work notifications and other information between friends. [**brien**]

We call spam to the not wanted electronic mail that contains information crafted to be delivered to a large number of recipients; without any of these explicitly ask

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

for receiving it. Spam is a huge problem to the everyday user that works using e-mail, they receive unwanted information about various products and companies that they have no interest in.

According to the ENISA (European Network and Information Security Agency) in 2009 the data on aborted SMTP connections and filtered e-mails seems to show that anti-spam measures are currently highly effective. Nearly 80% of SMTP connections are aborted, most of them due to blacklists. And from the accepted connections, nearly 80% are filtered out, mostly as spam. Thus, the percentage of delivered e-mail is only 4.4% of the total. The anti-spam measures are effectively filtering out vast amounts of spam, without allowing false positives to become a major problem. [**enisa**]

Multiple tools exist to fight this problem. There are several spam filters in the market that uses different methods for identifying and deleting junk mail. From Cormack's definition, A spam filter is an automated tool to recognize spam so as to prevent its delivery. The purpose of spam and spam filters are diametrically opposed: spam is effective if it evades filters, while a filter is effective if it recognizes spam. [**cormack**] We focused on the implementation of the Bayesian method, cover the code implementation, and tested on 2 different e-mail pools; one that is prefabricated and a real one that we have from a personal e-mail that contains spam in it.

In this document you will see the current status for spam filtering in general, we focus on the way the Bayesian method works and the results received with the implementation. In section 2 we discuss the existing methods and ways to filter spam e-mail. In section 3 we discuss the logic and algorithm for the Bayesian method. In section 4 we discuss how we got the testing e-mail to check the performance of our implementation. Section 5 shows the implementation of this algorithm. Sections 6 and 7 discuss the results and the conclusions that we got from the experiment.

2 Existing filters methods for spam

Currently there are multiple methods that we can use to filter junk e-mail. Some of these have a better performance than others, but ultimately its intent is to help the user to really concentrate on the e-mail that they have interest in. Almost all of the methods try to separate the e-mail in 2 categories; the legitimate e-mail and the spam e-mail. We present some of the most well known methods out there. While there are many methods, we only discuss a few.

2.1 The memory-based classifier

One of the learning methods, namely the memory-based classifier of TiMBL [**androutsopoulos**] . This is a popular choice because all the spam messages cover a very broad range of

topics. This suggest that memory-based algorithms, that uses a logic in which it attempts to learn unifying characteristics from similar previously received messages. The common feature of these methods is that they store all training instances in a memory structure, and use them directly for classification. We could use a memory structure like a multi-dimensional space defined by the attributes in the instance vectors. Each training instance is represented as a point in the space, the classification is a variant of the simple k-nearest neighbor (k-nn) algorithm.[**knn**]

2.2 Nave Bayesian classifier

The Bayes theorem and the theorem of the total probability, says that a document **D** with a vector $X = x_1, x_2, x_n$ composed from a set of mutually exclusive events, belongs to the category **C** said by the formula [**formula**]:

$$P(C = c|\mathbf{X{=}x}) = \frac{P(C = c).P(\mathbf{X{=}x}|C = c)}{\Sigma P(C = k).P(\mathbf{X{=}x}|C = k)} \quad ||k \in [spam, legitimate] \quad (1)$$

For multiple comparison purposes we used the work from Ion Androutsopoulos [**androutsopoulos**] to check the effectiveness of this method. We follow their metrics to ensure getting a better performance of the algorithm. Furthermore, we go more in depth about the metrics and the algorithm following the implementation of this particular method in the next section.

2.3 Chi by Degrees of freedom

The chi by degrees of freedom method is not usually used as a popular method for the e-mail filtering. According to Brien and Vogel [**brien**] it has the advantage of providing significance measures which could help eliminate false positives, a massive consideration when we're talking about filters. This method is often used in authorship identification, so it could be used successfully in detecting spam too. Recent studies [**brien**] say that millions of e-mails are work of only around 150 spammers in the world. We could use authorship identification methods to identify these textual fingerprints and eliminate a huge porportion of spam. The way it works is giving each file an indexed by n-grams, with frecuency counts. The n-grams can be of characters or words depending on specification. The method calculate the similarity value between the newly created files in terms of n-gram frecuencies.Calculating the chi-square test value by the number of its degress of freedom (number of n-grams minus one)

2.4 Black-White-Gray list

One of the simplest and most used methods for filtering spam since the beginning was the use of black and white list. As mentioned by Rosenberg [**lists**] we could have a new approach too, the gray list. The idea behind this method is to block any messages coming from a blacklist sender, allow any e-mail from Whitelist and evaluate to determine if we allow or block the e-mails coming from a graylist. The definition is:

- Black List. A list of the known spammers, this list contains the know systems that always send junk e-mail. Using this approach when an user receives a new message from one of the senders in this list it would go directly to the spam folder or it will be discarded.
- White List. On the other hand this is a list of the legit senders of e-mail. This list of senders will always be received in our inbox.
- Gray List. This is a new tricky one, everyday multiple companies try to send multiple e-mails to make their potential customers aware of discounts, deals, etc. When we receive an e-mail from this list it usually uses another method to check the content of the message, comparing this against the other 2 lists that only will check the sender e-mail.

3 The Naive Bayesian method

In this paper we are focusing on the Naive Bayes method, several authors tested this method for a long time in this particular area (spam filtering). We are trying to prove the efficiency and effectiveness mentioned by this papers.

3.1 The Naive Bayesian theory

First, if the Bayes Theorem would be implemented, you must understand what the theorem is about. In the probability field, the Bayes Tehorem [**pawlak**] was enunciated by Thomas Bayes in 1763, that expresses that the conditional probability of a random event A given B in terms of the distribution of conditional probability of the event B given A and the marginal probability distribution of only A. In more general terms, this theorem is important because of the importance that links probability of A given B with the probability of B given A. This theorem can help us indicate how we should change our subjestive probabilities when we receive aditional information in an experiment.

Tacking A_1, A_2, A_n as a group of mutually exclusive and exhaustive events, and the probability of each of them is bigger than zero. Let B be the event that any conditional probabilities are known$P(B|A_n)$. Then the probability of $P(A_n|B)$ is:

$$P(A_n|B) = \frac{P(B|A_n).P(A_n)}{P(B)} \quad ||where P(A_n) priori. P(A_n|B) posteori. \qquad (2)$$

3.2 The algorithm for the implementation

For the implementation of this method, we based on the implementation from Daniel Shiffman [**shiffman**]. Here is an example of a simplier version to show the way it works. You can refer to the mentioned sources to check the complete logic.

Naive Bayesian Spam Filter algorithm

Input: a set of e-mails to be classified A, a spam training file *SPAM*, a good e-mail training file *GOOD* **Output:** file name and classification of legit/spam to each

trainSpamWords(SPAM) Check the frequency of the words that will we classified as part of spam
trainLegitWords(GOOD) Check the frequency of the words that will we classified as part of legit e-mail
getProbabilities() Get each word probability based on their frequence
for *i := 1 to A.size* **do**
file = openfile(A_i) Open the file i and convert to one big string
splitFile(file) into tokens[]
Get top 10 most interesting A_i words

PROBA=getProbabilitybyBayes()
if PROBA ≥ 0.9 **then return** *true*
else return *false*
end for

4 The experimentation of the method

4.1 Getting the data collection

For testing purposes of the performance of the implementation, we choose to use some of the examples available for testing in the SPAM Archive [**archive**]. I used 3 different e-mail pools so we can compare the results and get an average for the performance. In the following table we share the details from the corpuses so we have an idea of the efficiency and efficacy of the implementation. The e-mail pool details are as follow:

Email Pool	Total Emails	Total Legit	Total Spam
Spam Archive 2013-01	1500	750	750
Spam Archive 2011	3000	1200	1800
Spam Archive 2012	4500	500	4000

4.2 Coding the solution

In the following table we mention the classes, methods and functionality of our Java implementation. This implementation is based on the one made by Danniel Shiffman [**shiffman**] based on the algorithm presented in Paul Grahams A Plan for Spam.[**graham**]

Java Classes

Class	Methods	Functionality
Lector.java	emptyContent()	This class is used to open the files for the training as the e-mails to be classified. The function is to open a stream for the file and empty its content in a string variable.
Palabra.java	calcBadProb(), calcGoodProb(), finalizeProb()	This class is used to storage each word to be identified in the files. It contains the number of times that appears in spam e-mails and in legit e-mails as the corresponding probability. It has as well the probability of the word being used in a spam.
Bayes.java	trainSpam(), trainGood(), analyze(), finalizeTraining()	This class is in the one we train our implementation to identify words associated with spam and with the legit e-mail. We calculate their probabilities too, finaly it cointains the analyze method to classify an e-mail depending on the words and frequency into spam or legit.
SpamClassifier.java	main()	This is the class with the main, it selects the files for training the algorithm and the folder in which all the files will be classified.

5 Testing and findings

For the testing we used 3 different e-mail pools as mentioned in section 4. Each of these files contain only spam so we added legit email to the folder gathered from our personal e-mail accounts. We ran through 3 scenarios, giving the different values for the total of spam and legit e-mail in each one to check the effectiveness.

These were our findings:

Email Collection	Total Emails	Total Legit	Total Spam	Legit Found	Spam found	Results
2013-01	1500	750	750	609	891	81.2 %
2011	3000	1200	1800	1333	1667	92.61 %
2012	4500	500	4000	1365	3135	78.37 %

From the testing, we did on the different pools some modification in the algorithm, these were made to make it more precise. Some of the notes that we took from this exercise were:

- The spam training file plays a pivotal role in the effectivenes of the method. With a first trainning file (cointaining only around 15 spam examples) the results were really low, the effectivenes was around 40% for the first run, after we gather a bigger (150 spam examples) with more diverse types we got the effectivenes up to 80-90 %. Getting good and diverse trainning files, both good and spam, determinates the effectivenes of the method and helps avoid false negatives.
- The interest of each word (The difference between the probability of the word being used in spam and 0.5) is important to check in each file. It will determine the factors to multiply the probability and classify the message.
- In the based-implementation which we took, it was based on the 15 first most frequent words of each e-mail. We found out that moving this number can alter the final result, if we open the range to 50, the probabilities of being spam for many files changes dramatictly. For my latest runs we tried to use the first 100 words, this only adds more correctnes.
- To reiterate, we based the implementation on Paul Graham's algorithm [**graham**]; he classified e-mail as "junk" when the probability is bigger than 0.9. In our test we consider that larger than 0.8 is spam. Modifying this interval will help us to fight the false negative.

6 Conclusion

As mentioned, spam is a great problem for the everyday user of e-mail service. Currently there are multiple methods and applications used to fight this issue. We discussed the Naive Bayesian method and how we can use it as an implementation to filter spam or "junk" e-mail. We discussed some other methods used for this task, like memory-based filters, black-list and white-list and the Chi by degrees of freedom method; to finally conclude that the Naive Bayesian method has multiple advantage over these. The algorithm can be controled in the trainning, the ranges and this will ultimately help us to fight false positives and false negatives.

We discused the implementation and the algorithm is based on, from our testing we conclude that this method is really flexible. We can modify a single element in the inputs and get a totally different efficiency and performance. The most important parts were getting a good trainning file for the spam and the legit email, this will help

give context to the algoithm and avoid getting false negatives. Other changes that we tried were moving the limits and ranges for the probabilities of each word, change the limits of the interesting words helping us fight the false positives.

Something to consider is that everyday multiple brands search new ways to send ”improved spam”. They find a way to send their messages with the filters not being able to detect them. We need to find new ways to fight this war. We believe this document can give the reader an idea of where and how to start.

References

1. CORMAC O'BRIEN AND CARL VOGEL *Spam Filters: Bayes vs Chi-squared; Letters vs Words.* 2002. Computational Linguistics Group and Computing and Language Studies. University of Dublin.
2. EUROPEAN NETWORK AND INFORMATION SECURITY AGENCY, *ENISA 2009 spam survey*, December 2009. US.
3. GORDON V. CORMACK *Email Spam Filtering: A Systematic Review* 2008. University of Waterloo, Ontario Canada.
4. ION ANDROUTSOPOULOS, GEORGIOS PALIOURAS, VANGELIS KARKALETSIS AND GEORGIOS SAKKIS *Learning to Filer Spam E-Mail: A comparison of a Naive Bayesian and a Memory-Based Approach* 2000. Software and Knoledge Engineering Laboratory, Athens, Greece.
5. KELLER, J.M.; GRAY, M.R.; GIVENS, J.A. *A fuzzy K-nearest neighbor algorithm* IEEE Transactions on , vol.SMC-15, no.4, pp.580,585, July-Aug. 1985
6. ION ANDROUTSOPOULOS, JOHN KOUTSIAS, KONSTANTINOS V CHANDRINOS, GEORGE PALIOURAS *An Evaluation of Naive Bayesian Anti-Spam Filtering* 2000. Software and Knoledge Engineering Laboratory, Athens, Greece.
7. J. ROSENBERG, C. JENNINGS *The Session Initiation Protocol (SIP) and Spam* January 2008.
8. ZDZISAW PAWLAK *Rough sets, decision algorithms and Bayes* European Journal of Operational Research, Volume 136, Issue 1, 1 January 2002, Pages 181-189, ISSN 0377-2217, http://dx.doi.org/10.1016/S0377-2217(01)00029-7.
9. DANIEL SHIFFMAN *Bayesian Filtering* 2011. http://shiffman.net/teaching/a2z/bayesian/.
10. PAUL GRAHAM *A plan for Spam* August 2002. Hackers and Painters. http://paulgraham.com/spam.html.
11. BRUCE GUENTER *Spam Archive* 2013. http://untroubled.org/spam/.

Chapter 3
Game Playing

Sliding Puzzle solved by a Modified A*.

Carlos Angulo

1 Introducción

Los juegos de Sliding Puzzle consisten en un tablero de NxN (siendo N un número natural), que contiene NxN -1 elementos, generalmente dispuestos en desorden, siendo el objetivo del juego ordenar los mismos en una secuencia dada, generalmente en orden ascendente comenzando por la fila superior desde el lado izquierdo hacia la derecha y progresivamente hasta la fila inferior, siendo este el caso de nmeros naturales consecutivos, pero también este tipo de juegos puede contener imágenes divididas, las cuales están al igual desordenadas.

2 Definición del problema y objetivos

Para resolver este tipo de juegos de una manera óptima se requiere conocer las jugadas posibles, siendo estas muy numerosas, las cuales pueden ser calculadas por medio de la fórmula (NxN)!/2, donde NxN el número total de lugares disponibles en el tablero, lo cual representa en el caso de tableros de 3x3 un total de 181,440 estados, y en el caso de un tablero de 4x4, se tienen un total de 10,461,394,944,000 de estados posibles, por lo que se requiere un algoritmo realmente eficiente para realizar la búsqueda de la solución a través de la exploración de las posibles jugadas hasta encontrara una jugada final.

En el caso de esta implementación se persigue resolver tableros de 3x3 por medio de un Algoritmo de Búsqueda A*. El algoritmo de Búsqueda A* consiste en recorrer las posibilidades de un grafo/ruta para hallar la ruta óptima (más corta) desde

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

un nodo/estado inicial hasta un nodo/estado meta tiendo en consideración el costo que implica el recorrer los diversos nodos más un estimado de cuanto nos falta para llegar a la meta, llamado heurístico.

En el caso de los Sliding Puzzles, existen dos heurísticos tradicionales, el Hamming y el Manhattan. El Hamming considera aquellos elementos del tablero que estn fuera de su lugar correspondiente y el costo representa la suma del número de elementos en desorden; en el caso de Manhattan es la suma de cada una de la distancia tanto horizontal como vertical de cada elemento a su posición correspondiente, para el caso de esta implementación y por razones de simplicidad se utilizará Hamming.

```
1  /* Pseudoc\'{o}digo de A* */
2  A_Star(inicio,meta) {
3  nodos_cerrados //Nodos ya explorados
4  nodos_abiertos //Nodos aun no evaluados
5  recorrido //Nodos anteriores al actual
6
7  costo_hacia_meta=0
8  costo_actual = costo_hacia_meta +
       costo_heur\'{i}stico_estimado(nodo_actual,nodo_meta)
9
10 while nodos_abiertos is not empty
11 if nodo_actual == nodo_meta
12       return
             reconsutrueye_camino(nodo_actual,nodo_inicio)
13
14 remover_de_nodos_abiertos (nodo_actual)
15 agregar_a_nodos_cerrados( nodo_actual)
16
17 recorrer_nodos_vecino(nodo_actual)
18       verificar_costo()
19       if vecino == meta
20             return
21             reconstruye_camino(nodo_vecino,nodo_inicio)
22
23 return
24
25 };
```

3 Metodología

Para evitar utilizar grandes cantidades de memoria el algoritmo implementado genera 5 expansiones de jugadas correspondientes, buscará en ellas la de menor costo posible, si en estos niveles no se encuentra la solución se expandirán más para hallar la solución, tomando en cuenta como el patron generador a aquel tablero que presente el menor costo posible.

En el caso de otros algoritmos de búsqueda estos expanden todos los nodos posibles, lo que requiere mucha memoria y tiempo de procesamiento, pero en este caso como se realiza de manera escalonada y esto no sucede, pero si tiene otras consideraciones importantes.

Entre las propuestas que se realizan para generar los movimientos posibles es dejar libremente que el tablero verifique hacia donde puede mover la pieza vacía, o bien de acuerdo a la información de una ficha dada con respecto a su posición y a donde se encuentra el lugar vacío, se generaran los posibles movimientos a realizar, esto puede resultar ineficiente, por lo que en esta implementación para resolver ese problema se busca primero en donde se encuentra la posición vacía y con esa información se determina cuantos son los movimientos posibles y cuáles son los mismos.

4 Problemas encontrados

Manejo de memoria por parte del compilador, se presentaron algunos problemas de segmentación de memoria, lo cual fue solucionado alocando la misma de manera estática.

Se debe implementar un mecanismo para no generar estados visitados previamente, en particular en este tipo de juegos, que fácilmente se generan jugadas idénticas a las previamente exploradas.

5 Resultados

Utilizando la implementación y utilizando como referencia el siguiente tablero se obtuvo una solución.

Los resultados obtenidos indican que la mayoría de los problemas resueltos se encuentran en la segunda iteración. Hay casos en los que no se logra resolver correctamente el tablero, aunque la implementación muestra que si se encontró el mismo, ocurren segmentaciones de memoria que no permiten corroborar que el resultado

sea correcto.

012
463
758

El recorrido que se generó para el tablero anterior fue el siguiente:

102 120 123 123 123 123
463 463 460 406 456 456
758 758 758 758 708 780

6 Conclusiones

El realizar un algoritmo de búsqueda eficiente requiere realizar muchas consideraciones para que el mismo funcione adecuadamente, el tratar de modificar un algoritmo ya bien definido si bien puede tener como resultados el aportar algo nuevo a una pequeña rama del conocimiento u optimizar el mismo para una implementacin o un problema muy específico puede conllevar que el problema se complique demasiado y que la optimización no sea tan significativa con relación al algorirtmo original en su forma pura.

6.1 Optimizaciones y trabajo futuro

El manejo de memoria en el algoritmo se realizó de manera estática, podría realizarse de manera dinámica, podrían utilizarse otros tipos de datos para representar los datos para que la memoria utilizada sea menor.

En el caso general también podría expandirse la implementación para resolver tableros más grandes con N igual o mayor a 4.

6.2 Pensamientos finales

El resolver problemas que para el hombre pueden parecer en cierto punto triviales, el tratar en ensearle a una máquina a hacerlo puede resultar una tarea muy compleja y más demandante que el problema mismo, sin embargo los resultados son muy

alentadores.

References

1. PRINCENTON, http://www.cs.princeton.edu/courses/archive/fall12/cos226/assignments/8puzzle.html,

Algorithm for Pokemon battle game using MinMax

Gustavo Montes

Abstract Pokemon (Pocket Monster) is a role-play game with elements of strategy that revolves around the catching and battling of Pokemon. Initially released on 1998, the original games had about 151 of these creatures. Today, this number has been increased up to 649 while adding more elements of battling strategy. The game has become so popular that now it has its own metagame. This article takes elements of this metagame to build an AI that can compete with skilled players.

1 Introduction

The main idea of the game is that the player start with one Pokemon, as the game flows; the player can catch wild Pokemon or battle them to raise levels and learn new moves. The player also encounters other in-game trainers that he must defeat in order to progress on the game. Additionally, the player can also battle against other players. This article will cover this last element of battling and will implement and effective algorithm for 1vs1 battles.

2 Game Mechanics

Each Pokemon has attributes and qualities that differentiate it from other Pokemon and make it unique.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

2.1 Name

The name identifies the Pokemon. The complete list of Pokemon used for this project is on the appendix of this article.

2.2 Typing

There are 17 different types; each type has its own weaknesses and strengths against the others, the rules to determine this are as follows.

- A type that is super effective against other will deal 2x damage.
- A type that is not very effective against other will deal 0.5x damage.
- A type that has immunity against other will take 0x damage.
- If those conditions arent met, the damage is not modified.

The below table is a representation of the different types and how they fare against each other.

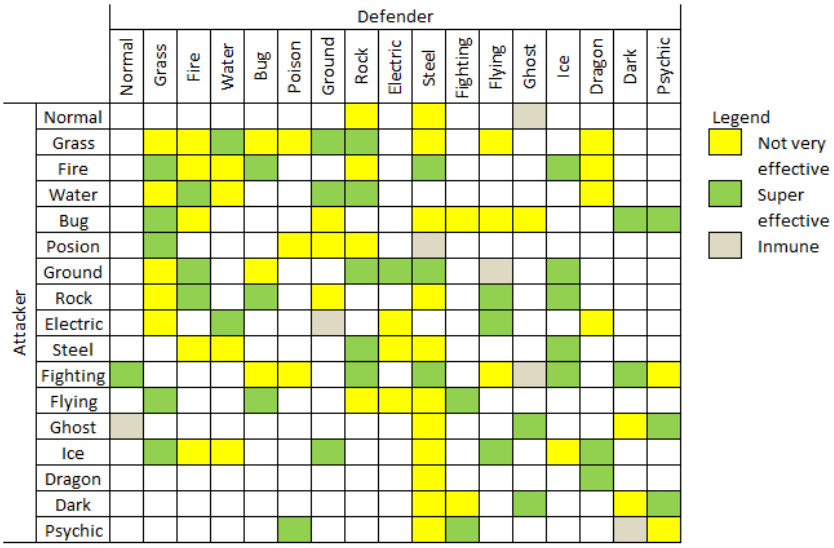


Fig. 1: A Pokemon can have 1 or 2 types, that is, if its typing is double weak or double resistant, it can take 4x or 0.25x damage (e.g. a Pokemon with Dragon/Flying typing will take 4x damage from an Ice attack)

2.3 Moves

A move is what a Pokemon uses to attack the opponent, it can learn up to 4 moves at the same time but it can only use one per turn. Every move has the following attributes:

- **Name:** The name of the move
- **PP (Power Points):** Is the number of times a Pokemon can use a move during battle, if the PP reaches 0, the Pokemon wont be able to use that move anymore during that battle. If a Pokemon has used all the moves it has available, it will start to inflict damage to itself until it faints.
- **Power:** The base damage of the move.
- **Accuracy:** The base accuracy of the move is the probability of the move to hit.
- **Type:** The type of the move, determines against what Pokemon is super effective and what not. If a Pokemon uses a move that has the same type as it, the base damage is multiplied by 1.5.
- **Effect:** Each move may have or not an effect, effects are particular situations that occur after a move is used (e.g. A move that has 10% chance to reduce the evasion of the target).
- **Category:** There are 3 categories of moves: Physical moves, Special moves and Status moves. Physical moves use the Attack of the Pokemon to inflict damage, Special moves use the Special Attack of the Pokemon and finally, the Status moves, dont use a base stat, but it purpose is to inflict an effect. The complete list of moves and its effects is on the appendix of this article.

  The complete list of moves used for this project is listed on the Appendix.

2.4 Base Stats

Base Stats are values that determine the strength of a Pokemon, they are intrinsic to it and never change. Their values round between 1 and 255.

- **Base HP (Hit Points):** It is the quantity of damage a Pokemon can take before fainting.
- **Base Attack:** Determines how much damage a Pokemon deals by using a physical attack.
- **Base Defense:** Determines how much damage a Pokemon receives when hit by a physical attack.
- **Base Special Attack:** Determines how much damage a Pokemon deals by using a special attack .
- **Base Special Defense:** Determines how much damage a Pokemon receives when hit by a special attack.
- **Base Speed:** Determines how quick the Pokemon can act, Pokemon with higher speed will move before a lower one under normal conditions.

2.5 Individual Value (IV)

Each Base Stat also has Individual Values, IVs are numbers between 0 and 31 that are assigned randomly when the Pokemon is created and cannot be changed.

2.6 Effort Value (EV)

Effort Values are points that the Pokemon gains when battling, there are EVs for every Base Stat and they can sum up to 255, but a Pokemon cannot have more than 510 EV's in total. For example, a Pokemon can have 252 points in Attack and Speed and 6 points in HP, but it wont be able to add points on any other stat anymore. They can be changed at any time.

2.7 Level

The level of the Pokemon determines how it Current Stats are, they are maximised at level 100 and minimized at level 1.

2.8 Nature

There are 25 Natures in Pokemon. They are modifiers that affect the growth rate of 2 stats of the Pokemon, ultimately increasing one of its stats by 10% and decreasing another by 10%. There are 5 natures that doesn't affect the growth of the stats. The complete list of nature is on the appendix of this document.

2.9 Current Stats

Though Base Stats determines the strenght of the Pokemon, they aren't used in battle. Instead, Current Stats are calculated using the above elements and are the ones that are actually used on battle.

The formula to calculate the HP is as follows:

$$HP = \frac{(IV + 2Base + \frac{EV}{4} + 100) * Level}{100} + 10$$

And the rest of the stats use the following:

$$Stat = [\frac{(IV + 2Base + \frac{EV}{4}) * Level}{100} + 5] * Nature$$

The result is always rounded down. In the case of the non HP stats, the result is rounded down before the Nature modifier is applied.

2.10 In-battle Stats

There are stats that are only used on battle, but they don't have use outside of it.

- **Evasion:** Determines the percent chance an opponents move will miss. Starts with 100% at the beginning of the battle and can be reduced or increased.
- **Accuracy:** Determines the percent chance an attackers move will hit. Starts with 100% at the beginning of the battle and can be reduced or increased.

  The probability that a move will hit is determined as follows:

$$P = \frac{MoveAcc * Accuracy}{Evasion}$$

Where:
MoveAcc: The accuracy of the move.
Accuracy: The current accuracy of the user.
Evasion: The current evasion of the target.
P: The probability of the move hitting, if P is greater thant 1, the move will surelly hit.

2.11 Ability

In addition to those stats, a Pokemon can have one or more abilities, abilities are special characteristics from the Pokemon that provides specific advantages over other Pokemon (e.g. The ability Levitate provides immunity against Ground moves), but it can use only one at time. The complete list of abilities is on the appendix of this article.

2.12 Status

Status ailments affect a Pokemons ability to battle; we will use only the most common of them for this article.

- **Burn:** Damage dealt by a Pokemons physical moves and loses 1/8 of its HP at the end of each turn. Fire-type Pokemon cannot be burned.
- **Paralisys:** There is a 25% chance this Pokemon wont attack that turn. Additionally, its speed is reduced to 25% of its previous value. Ground-type Pokemon cant be paralyzed.
- **Poison:** Causes the Pokemon to lose 1/8 of its HP every turn. Steel and Poison-type Pokemon cannot be poisoned.
- **Bad Poison:** Its damage begins at 1/16 and grows an additional 1/16 every turn. Steel and Poison-type Pokemon cannot be poisoned.
- **Sleep:** Pokemon cannot attack. Last for a randomly duration of 1 to 3 turns.
- **Freeze:**The Pokemon cannot attack. There is a 20% chance to be cured every turn.

3 Battling

3.1 The Damage Formula

The key element of every battle is the damage dealed by a Pokemon. The Damage formula is itself a huge list of steps, conditions and multiplications, provided that the move will hit. Here is the formula explained for this project:

$$BaseDamage = \frac{(\frac{(2Level)}{5} + 2) * BasePower * [Sp]Atk}{[Sp]Def * 50 + 2}$$

1. If the move is a non-attacking one, apply the effect and end the function.
2. Determine if the hit will be critical.
3. Apply the Typing modifiers.
4. Determine if the damage will be Physical or Special and apply Attacking and Defending modifiers, except if the hit is critcal, in this case, the Defending modifiers are ignored.
5. Calculate the base damage with the above formula.
6. If critical hit, multiply the base damage by 2.
7. Apply Ability modifiers.
8. The base damage if segmented into 16 values, each one going from 0.85x to 1.0x the original value.
9. From the list, select a random value and this will be the damage dealed by the Pokemon.

10. If the damage is greater than 0 and has effect, apply.

3.2 The flow of the battle

The battle starts with both Pokemon being sent out. From this moment, each Pokemon has 4 moves to choose from, when both players have choosen a move, the Pokemon will attack each other, moving first the one who has better speed or the one whom move has higher priority. After the attacks have been done, the turn ends. They keep doing this until one of the Pokemon faints.

The following can be seen as the representation of the flow of a battle.

```
Turn 1
Gengar used Shadow Ball!
The opposing Quagsire lost 73% of its health!
The opposing Quagsire's Special Defense fell!
The opposing Quagsire used Waterfall!
Gengar lost 62% of its health!

Turn 2
Gengar used Shadow Ball!
The opposing Quagsire lost 33% of its health!
The opposing Quagsire fainted!
```

4 Implementation

4.1 MiniMax and AlphaBeta

Consider the case of 2 players, one of which is MAX and the other one MIN. MAX always move first, and they take turns battling each other until the game is over. When the game finishes, points are awarded to the winning player. A game can be formally described with the following elements:

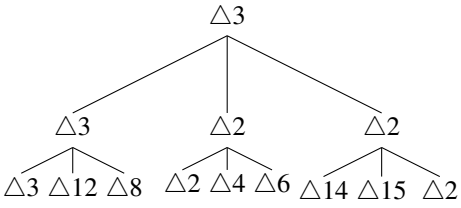
- **Initial State:** The state of the game before any player makes a move and who moves first.
- **Set of Operators:** What are the legal moves a player can make.
- **Terminal State:** Determines when the game is over.
- **Utility Function:** Gives a numeric value to the outcome of the game, it is negative if MIN wins or positive if MAX does.

MAX has to find a way where it reaches a Terminal State where it wins, and then make the necessary moves in order to reach that state. But MIN does the opposite

and will make any necessary move that would make MAX lose.

The minimax algorithm is used to determine the optimal strategy for MAX, and thus decide what the best first move is. The algorithm consists of 5 steps:

1. Generate the whole game tree.
2. Use the utility function to get the score of every Terminal State.
3. Use the score of the Terminal States to get the score of the State that is one level above. MIN will always use the lowest score and MAX the highest.
4. Repeat last step from the leafs towards the root, one layer per time.
5. There will be a moment where you reach the node, at that point, MAX chooses the highest value and thats the move it will make.

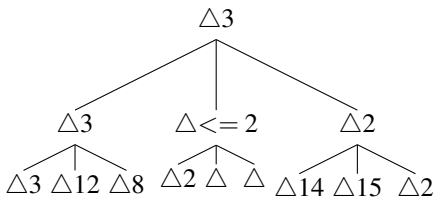


However, minimax assumes that the program has the time and resources to deploy the whole tree, and in most of the cases, this is not possible, as the number of nodes generated are immense. For this, you can implement a cut-off to tell the program how deep you want to search in the tree, and apply an evaluation function depending on what is the state of the game at that node.

But even with a good evaluation function and a cut-off, there are problems that still have several issues with the performance. Lets assume that you have an initial state, and that each player has 16 available moves and a deep or cut-off of 10, the number of explored nodes go around $10.9 * 10^{11}$, programs will still take a lot of time just to make a single move.

There is another enhancement to this algorithm that still finds the optimal path without the necessity to look at all the nodes of the tree. This is called **prunning** the tree and the technique is called **alpha-beta prunning**.

The general principle is this. Consider a node n somewhere in the tree, such that the Player has a choice of moving to that node. If the Player has a better choice either at the parent node of n, or at any choice point further up, then n will never be reached in actual play. So once we have found out enough about n (by examining some of its descendants) to reach this conclusion, we can prune it. The following is a representation of a tree searched by alpha-beta prunning.



4.2 Algorithm for Pokemon

With the alpha-beta prunning approach, I designed an algorithm that works similarly but also has the condition of who move first and if its the begining of the turn. The change here is that the opponent is MAX and the algorithm will try to find the best move for it, and in order to simplify things, this function is called after the user chooses a move, meaning that it doesn't have to look for all actions the user could make, at least for the top level of the tree.

```
function alphaBeta(node, alpha, beta, depth, isPlayer, beginOfTurn)
  if depth = 0 or node is terminal then
    return score of the node;
  if beginOfTurn is true then
    if isPlayer is true then
      value = -Infinity;
      foreach child in the node
        attack(node.user, node.ai);
        value = max(value, alphaBeta(node, alpha, beta, depth, false, false));
        if value >= beta then
          return value;
      return value;
    else
      value = Infinity;
      foreach child in the node
        attack(node.ai, node.user);
        value = min(value, alphaBeta(node, alpha, beta, depth, true, false));
        if value <= alpha then
          return value;
      return value;
  else
    if isPlayer is true then
      value = -Infinity;
      foreach child in the node
        attack(node.user, node.ai);
        value = max(value, alphaBeta(node, alpha, beta, depth-1, false, true));
        if value >= beta then
```

```
          return value;
      return value;
    else
      value = Infinity;
      foreach child in the node
        attack(node.ai, node.user);
        value = min(value, alphaBeta(node, alpha, beta, depth-1, true, true));
        if value <= alpha then
          return value;
      return value;
end alphaBeta;
```

The score of the node goes between -100 and 100. And its determined by the difference of HP in percentage of both Pokemon. A score of -100 means that the opponent defeated the user without loosing any HP and viceversa.

The algorithm will stop whenever the *depth* reaches 0 or one of the Pokemon faints, note that depth will only decrease when *beginOfTurn* is false, that is, the end of the turn.

Elements such as chance of hitting or if the Pokemon is faster are determined between the damage calculator. For example a move with a BP of 120 and Accuracy of 80 will have a weighted BP by multiplying both.

$$WeightedBP = BP * Acc$$
$$WeightedBP = 120 * 0.8$$
$$WeightedBP = 96$$

In order to avoid even more unnecesary search, there is a condition that if a Pokemon has a move that doesn't affect the oponent (merely trough typing or abilities), it wont be taken as an eligible action.

5 Results

5.1 Scenario 1: Dragonite vs Heatran

```
Dragonite Moves: DragonDance, CloseCombat, FirePunch, Earthquake
Heatran Moves: FireBlast, LavaPlum, EarthPower, Toxic
```

For the first case, Dragonite just have to select Earthquake and doesn't matter what Heatran chooses because Dragonite is faster and will defeat it on 1 hit.

```
Turn 1
```

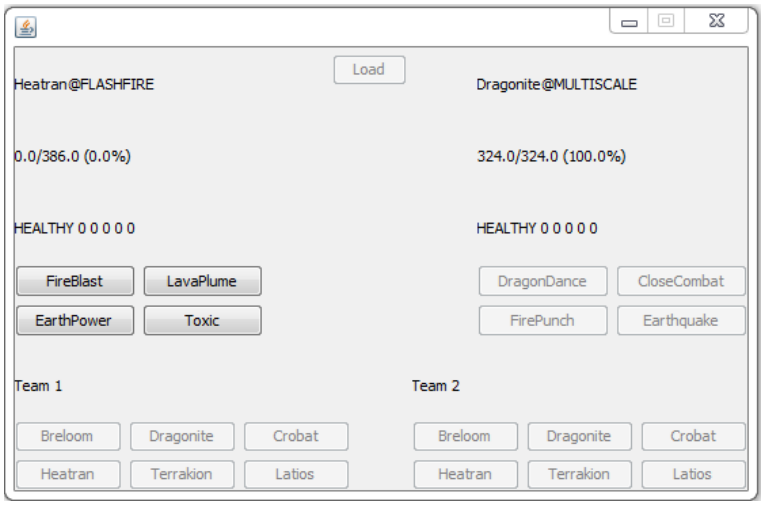


Fig. 2: Heatran vs Dragonite

```
value: 100.0 Heatran: FireBlast Dragonite: Earthquake
value: 100.0 Heatran: LavaPlume Dragonite: Earthquake
value: 100.0 Heatran: Toxic Dragonite: Earthquake
Nodes searched: 3
action FireBlast
```

All options Heatran has have the same result because it wont be able to inflict damage, notice that Earth Power is omited as it is a Ground move and Dragonite is part Flying. But what if Heatran was faster than Dragonite?

```
Turn 1
value: 85.19 Heatran: FireBlast Dragonite: Earthquake
value: 90.12 Heatran: LavaPlume Dragonite: Earthquake
value: 100.0 Heatran: Toxic Dragonite: Earthquake
Nodes searched: 508
action FireBlast
```

The result are almost the same, the only difference is that Dragonite will not take damage only if Heatran chooses Toxic and the worst scenario for him is when Heatran chooses FireBlast leaving him at 85.19%. Notice that the nodes searched is higher than the previous case due to Heatran moving first, hence, the algorithm didn't cut the first level of the tree that would be the static move choosen by Dragonite.

5.2 Scenario 2: Dragonite vs Latios

```
Dragonite Moves: DragonDance, CloseCombat, FirePunch, Earthquake
Latios Moves: DracoMeteor, Surf, Thunderbolt, Psychic
```

Fig. 3: Dragonite vs Latios

How does Dragonite fare against a quicker and stronger opponent?

```
Turn 1
value: -65.56 Latios: DracoMeteor Dragonite: DragonDance
Nodes searched: 307
action DracoMeteor
Turn 2
value: -48.34 Latios: DracoMeteor Dragonite: CloseCombat
Nodes searched: 40
action DracoMeteor
```

Dragonite loses no matter what he does, Latios just have to spam Draco Meteor or any combination of Draco Meteor / other move. On the other hand, if I don't play Latios optimally, the best option for Dragonite is to use Dragon Dance and then spam Close Combat because after one setup, he will be faster and will hit harder.

```
Turn 1
value: 65.56 Dragonite: DragonDance Latios: Surf
Nodes searched: 339
action DragonDance
```

```
Turn 2
value: 100.0 Dragonite: DragonDance Latios: Surf
value: 48.34 Dragonite: CloseCombat Latios: Surf
Nodes searched: 182
action CloseCombat
Turn 3
value: 48.34 Dragonite: DragonDance Latios: Surf
value: -62.65 Dragonite: CloseCombat Latios: Surf
Nodes searched: 41
action CloseCombat
```

5.3 Scenario 3: Crobat vs Breloom

```
Crobat Moves: SuperFang, CrossPoison, BraveBird, Toxic
Breloom Moves: LowSweep, MachPunch, Spore, SwordDance
```

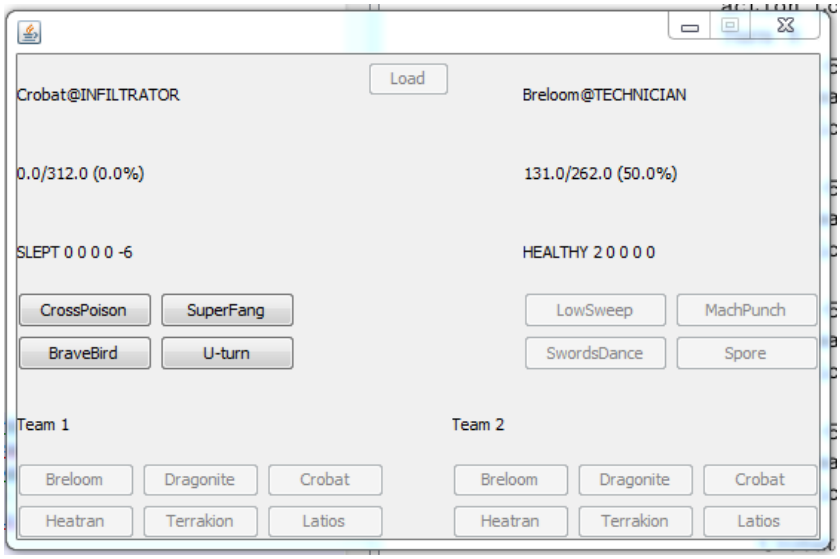


Fig. 4: Crobat vs Breloom

The following scenario is one of the most interesting cases to see, Crobat has a clearly advantage over Breloom due to typing, speed and attacks. One single hit from Crobat and Breloom is death, but what happens if Crobat makes a mistake and doesn't kill Breloom?. See the following log:

```
Turn 1
```

```
value: 5.770000000000003 Breloom: LowSweep Crobat: SuperFang
Nodes searched: 5100
action LowSweep
Turn 2
value: 77.56 Breloom: LowSweep Crobat: CrossPoison
value: -27.88 Breloom: Spore Crobat: CrossPoison
Nodes searched: 5049
action Spore
Turn 3
value: -39.1 Breloom: LowSweep Crobat: CrossPoison
value: -50.0 Breloom: SwordsDance Crobat: CrossPoison
Nodes searched: 17268
action SwordsDance
Turn 4
value: -50.0 Breloom: LowSweep Crobat: CrossPoison
Nodes searched: 5208
action LowSweep
Turn 5
value: -50.0 Breloom: LowSweep Crobat: CrossPoison
Nodes searched: 2541
action LowSweep
Turn 6
value: -50.0 Breloom: LowSweep Crobat: CrossPoison
Nodes searched: 1319
action LowSweep
Turn 7
value: -50.0 Breloom: LowSweep Crobat: CrossPoison
Nodes searched: 548
action LowSweep
Turn 8
value: -50.0 Breloom: LowSweep Crobat: CrossPoison
Nodes searched: 166
action LowSweep
```

Crobat used SuperFang that only took half life from Breloom, Breloom took the opportunity and retaliated with Low Sweep. Next turn, Breloom is faster because of Low Sweep decreasing Crobat speed, from there, Breelom puts Crobat to sleep so it wont be able to attack, and setups a Sword Dance. All what Breloom has to do is to spam Low Sweep until Crobat is death. Note: Sleep can only last at worst 3 turns, but for this project, I simplified some variables and sleeping last forever.

6 Conclusions

There are still a lot of growing opportunities for this project. But due to the nature of the game and the immense quantity of variables such as moves, pokemons, weather effects, status moves real behavior, etc. I had to simplify things and only covered a small portion of the game.

The project was focused to 1vs1 battles only, but there is the option to grow and implement a full 6vs6 battle, sure, there will be more limitations with the algorithm due to number of nodes increasing exponentially. For example, for this project, a Pokemon had only 4 moves to choose from, but with a full team, it would have 9 options to choose from!, with only 3 turns the tree could be expanded with more than 500,000 nodes!.

A better evaluation function could help to past this problem, there is more to Pokemon than just the remaining HP of it. Sometimes, its better to sacrifice something in order to find a setup opportunity and from there, a single Pokemon can sweep the whole team by itself. But identifying those opportunities requires a lot of understanding of the game and experience with battles.

Another approach would be to actually implement a battling strategy for the AI, even if they weren't covered by this article, there are a lot of strategies which consist on countering certain teams, such as Stall teams, Hyper Offense, Balanced, etc.

But the most important thing is to demonstrate that the MinMax algorithm can be useful to find optimal paths on a game where everything can happen.

7 Appendix

7.1 Abilities

Ability	**Effect**
Multiscale	While the wielder is at 100% HP, all damage from attacks is halved.
Technician	Increases the Base Power of moves with 60 or less by 50%.
Levitate	The wielder is immune to Ground-type moves.
Flash Fire	Grants immunity to Fire-type moves and increases the power of Fire-type moves by 50% when hit by a Fire-type move.
Justified	Raises Attack one stage when hit by a Dark-type move.
Infiltrator	The wielder bypasses the foe's Light Screen, Reflect, Mist, and Safeguard

7.2 Pokemon

Pokemon	**Typing**	**HP**	**Atk**	**Def**	**SpA**	**SpD**	**Spe**
Dragonite	Dragon/Flying	91	134	95	100	100	80
Latios	Dragon/Psychic	80	90	80	130	110	110
Terrakion	Rock/Fighting	91	129	90	72	90	108
Breloom	Grass/Fighting	60	130	80	60	60	70
Heatran	Steel/Fire	91	90	106	130	106	77
Crobat	Poison/Flying	85	90	80	70	80	130

7.3 Moves

Move	Type	PP	Power	Category	Chance	Effect
Dragon Dance	Dragon	20	–	Effect	100	Raises user Atk and Spa by 1 stage
Earthquake	Ground	10	100	Physical	100	–
Fire Punch	Fire	10	75	Physical	100	10% chance to burn the target
Psychic	Psychic	10	90	Special	100	10% chance to lower SpD by 1 stage
Close Combat	Fighting	5	120	Physical	100	Lowers user Def and SpD by 1 stage
Earth Power	Ground	10	90	Special	100	10% chance to lower target SpD by 1 stage
Draco Meteor	Dragon	5	140	Special	90	Lower user SpA by 2 stages
Stone Edge	Rock	5	100	Physical	80	–
Low Sweep	Fighting	20	60	Physical	100	Lower target Spe by 1 stage
Lava Plume	Fire	15	80	Special	100	30% chance to burn the target
Surf	Water	15	95	Special	100	–
X-Scissor	Bug	15	80	Physical	100	–
Fire Blast	Fire	5	120	Special	85	10% chance to burn the opponent
Thunderbolt	Electric	15	95	Special	100	10% chance to paralze the opponent
Swords Dance	Normal	30	–	Effect	100	Raises user Atk by 2 Stages
Brave Bird	Flying	10	120	Physical	100	The user receives recoil damage equal to 1/3 of the damage dealt
U-Turn	Bug	20	70	Physical	100	The user leaves the battle and a teammate enters on its place
Super Fang	Normal	20	1	Physical	90	Damage is equal to half target HP
Cross Poison	Poison	10	70	Physical	100	Damge is equal to half target HP

7.4 Natures

Name	**+**	**-**
Adamant	Atk	SpA
Bashful		
Bold	Def	Atk
Brave	Atk	Spe
Calm	SpD	Atk
Careful	SpD	SpA
Docile		
Gentle	SpD	Def
Hardy		
Hasty	Spe	Def
Impish	Def	SpA
Jolly	Spe	SpA
Lax	Def	SpD
Lonely	Atk	Def
Mild	SpA	Def
Modest	SpA	Atk
Naive	Spe	SpD
Naughty	Atk	SpD
Quiet	SpA	Spe
Quirky		
Rash	SpA	SpD
Relaxed	Def	Spe
Sassy	SpD	Spe
Serious		
Timid	Spe	Atk

References

1. S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd Edition, 2009.
2. Edwards, D.J. and Hart, *The Alphabeta Heuristic (AIM-030)*. Massachusetts Institute of Technology, 4 December 1961 to 28 October 1963
3. Knuth, D. E., and Moore, R. W. *An Analysis of AlphaBeta Pruning*. Artificial Intelligence, 1975
4. The Complete Damage Formula for Black & White, `http://www.smogon.com/bw/articles/bw_complete_damage_formula`
5. Pokemon, `http://www.smogon.com/bw/pokemon/`
6. Types, `http://www.smogon.com/bw/types/`
7. Moves, `http://www.smogon.com/bw/moves/`
8. Abilities, `http://www.smogon.com/bw/abilities/`
9. Natures, `http://www.smogon.com/bw/natures/`

10. Status, http://www.smogon.com/bw/articles/bw_status
11. Stats, http://bulbapedia.bulbagarden.net/wiki/Stats#Permanent_stats

A* Pathfinding Algorithm used in videogames

Erik Castro Estrada

Abstract The main objective of this work is to implement the knowledge, algorithms and artificial intelligence techniques studied in class Intelligent Systems. This paper describes the application of search algorithms applied to a video game focusing on the use of A * search algorithm as a controller that could play a version of Super Mario Bros as well as possible. The performance of this algorithm is analyzed in the game Mario Bros. This work is intended to demonstrate the usage of a simple pathfinding algorithm in a video game and seeks to be a starting point to more advanced artificial intelligence techniques and its implementation in videogames.

1 Introduction

Platform games can be defined as games where the player controls a character/avatar, usually with humanoid form, in an environment characterized by differences in altitude between surfaces (platforms) interspersed by holes/gaps. The character can typically move horizontally (walk) and jump, and sometimes perform other actions as well; the game world features gravity, meaning that it is seldom straight forward to negotiate large gaps or altitude differences. Most commercial platform games incorporate little or no AI. The main reason for this is probably that most platform games are not adversarial; a single player controls a single character who makes its way through a sequence of levels, with his success dependent only on the player's skill. The obstacles that have to be overcome typically revolve around the environment (gaps to be jumped over, items to be found etc) and NPC enemies; however, in most platform games these enemies move according to preset patterns or simple homing behaviours. Though apparently an under-studied topic, artificial intelligence for controlling the player character in platform games is certainly not without interest. From a game development perspective, it would be valuable to

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

be able to automatically create controllers that play in the style of particular human players. This could be used both to guide players when they get stuck and to automatically test new game levels and features as part of an algorithm to automatically tune or create content for a platform game. From an AI and reinforcement learning perspective, platform games represent interesting challenges as they have high-dimensional state and observation spaces and relatively high-dimensional action spaces, and require the execution of different skills in sequence. Further, they can be made into good testbeds as they can typically be executed much faster than real time and tuned to different difficulty levels.

2 Infinite Mario Bros

For this work a modified version of Markus Persson's Infinite Mario Bros is used, which is a public domain clone of Nintendo's classic platform game Super Mario Bros. The original Infinite Mario Bros is playable on the web, where Java source code is also available[1]. The gameplay in Super Mario Bros consists in moving the player-controlled character, Mario, through two-dimensional levels, which are viewed sideways. Mario can walk and run to the right and left, jump, and (depending on which state he is in) shoot fireballs. Gravity acts on Mario, making it necessary to jump over holes to get past them. Mario can be in one of three states: Small, Big (can crush some objects by jumping into them from below), and Fire (can shoot fireballs). The main goal of each level is to get to the end of the level, which means traversing it from left to right. Auxiliary goals include collecting as many as possible of the coins that are scattered around the level, finishing the level as fast as possible, and collecting the highest score, which in part depends on number of collected coins and killed enemies. Complicating matters is the presence of holes and moving enemies. If Mario falls down a hole, he loses a life. If he touches an enemy, he gets hurt; this means losing a life if he is currently in the Small state. Otherwise, his state degrades from Fire to Big or from Big to Small. However, if he jumps and lands on an enemy, different things could happen. Most enemies (e.g. goombas, cannon balls) die from this treatment; others (e.g. piranha plants) are not vulnerable to this and proceed to hurt Mario; finally, turtles withdraw into their shells if jumped on, and these shells can then be picked up by Mario and thrown at other enemies to kill them. Certain items are scattered around the levels, either out in the open, or hidden inside blocks of brick and only appearing when Mario jumps at these blocks from below so that he smashes his head into them. Available items include coins, mushrooms which make Mario grow Big, and flowers which make Mario turn into the Fire state if he is already Big.

[1] http://www.mojang.com/notch/mario/

2.1 Automatic Level Generation

While implementing most features of Super Mario Bros, the standout feature of Infinite Mario Bros is the automatic generation of levels. Every time a new game is started, levels are randomly generated by traversing a fixed width and adding features (such as blocks, gaps and opponents) according to certain heuristics. The level generation can be parameterized, including the desired difficulty of the level, which affects the number and placement of holes, enemies and obstacles. In the modified version of Infinite Mario Bros we can specify the random seed of the level generator, making sure that any particular level can be recreated by simply using the same seed. Unfortunately, the current level generation algorithm is somewhat limited; for example, it cannot produce levels that include dead ends, which would require back-tracking to get out of.

2.2 Challenges for AIs (and humans)

Several features make Super Mario Bros particularly interesting from an AI perspective. The most important of these is the potentially very rich and high-dimensional environment representation. When a human player plays the game, he views a small part of the current level from the side, with the screen centered on Mario. Still, this view often includes dozens of objects such as brick blocks, enemies and collectable items. The static environment (grass, pipes, brick blocks etc.) and the coins are laid out in a grid (of which the standard screen covers approximately 22 * 22 cells), where moving items (most enemies, as well as the mushroom power-ups) move continuously at pixel resolution. The action space, while discrete, is also rather large. In the original Nintendo game, the player controls Mario with a D-pad (up, down, right, left) and two buttons (A, B). The A button initiates a jump (the height of the jump is determined partly by how long it is pressed); the B button initiates running mode and, if Mario is in the Fire state, shoots a fireball. Disregarding the unused up direction, this means that the information to be supplied by the controller at each time step is five bits, yielding $2 \exp 5 = 32$ possible actions, though some of these are nonsensical (e.g. left together with right). Another interesting feature is that there is a smooth learning curve between levels, both in terms of which behaviours are necessary and their required degree of refinement. For example, to complete a very simple Mario level (with no enemies and only small and few holes and obstacles) it might be enough to keep walking right and jumping whenever there is something (hole or obstacle) immediately in front of Mario. A controller that does this should be easy to learn. To complete the same level while collecting as many coins as possible present on the same level likely demands some planning skills, such as smashing a powerup block to retrieve a mushroom that makes Mario Big so that he can retrieve the coins hidden behind a brick block, and jumping up on a platform to collect the coins there and then going back to collect the coins hidden under it. More advanced levels, including most of those in the original Super Mario Bros game, re-

```
// always the same dimensionality 22x22
// always centered on the agent
public byte[][] getCompleteObservation();
public byte[][] getEnemiesObservation();
public byte[][] getLevelSceneObservation();
public float[] getMarioFloatPos();
public float[] getEnemiesFloatPos();
public boolean isMarioOnGround();
public boolean mayMarioJump();
```

Fig. 1: The Environment Java interface, which contains the observation, i.e the information the controller can use to decide which action to take.

quire a varied behaviour just to complete. These levels might include concentrations of enemies of different kinds which can only be passed by timing Mario's passage precisely; arrangements of holes and platforms that require complicated sequences of jumps to pass; dead ends that require backtracking; and so on.

3 Mario API

For this project we are using a modified Infinite Mario Bros the API enables it to be easily interfaced to learning algorithms and AI controllers. The modifications included removing the real-time element of the game so that it can be stepped forward by the learning algorithm, removing the dependency on graphical output, and substantial refactoring. Each time step, the controller receives a description of the environment, and outputs an action. The resulting software is a single-threaded Java application that can easily be run on any major hardware architecture and operating system, with the key methods that a controller needs to implement specified in a single Java interface file (see figure 1). Figure 2 shows the size and layout relative to Mario of the sensor grid. For each grid cell, the controller can choose to read the complete observation (using the *getCompleteObservation()* method) that returns an integer value representing exactly what occupies that part of the environment (e.g. a section of a pipe), or one of the simplified observations (*getEnemiesObservation()* and *getLevelSceneObservation()*) that simply returns a zero or one signifying the presence of an enemy or an impassable part of the environment.

4 A* Search

As the goal is to complete a level as fast as possible, without regarding points or other bonuses, the problem can be seen as a path optimization problem: What is the quickest route through the environment that doesn't get Mario killed? A* search is

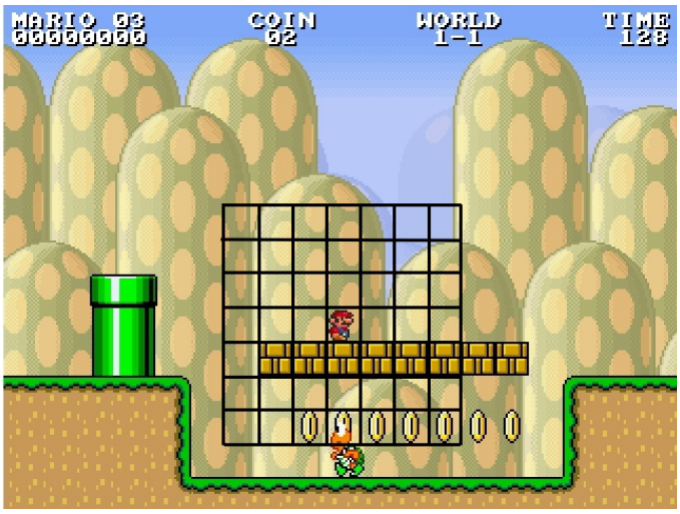


Fig. 2: Visualization of the granularity of the sensor grid. The top six environment sensors would output 0 and the lower three input 1. All of the enemy sensors would output 0, as even if all 49 enemy sensors were consulted none of them would reach all the way to the body of the turtle, which is four blocks below Mario. None of the simplified observations register the coins, but the complete observation would. Note that larger sensor grids can be used (up to 22 * 22), if the controller can handle the information.

a well known, simple and fast algorithm to find shortest paths which seems perfect to solve this issue. As it turns out, A* search is fast and flexible enough to find routes through the generated levels in real-time. The implementation process can be separated into three phases: (a) building a physics simulation including world states and object movement, (b) using this simulation in an A* planning algorithm, and (c) optimizing the search engine to fulfill real-time requirements with partial information.

a) Simulating the Game Physics: Due to being open source, the entire physics engine of Infinite Mario Bros is directly accessible and can be used to simulate future world states that correlate very closely with the actual future world state. This was achieved by copying the entire physics engine to the controller and removing all unnecessary parts, such as rendering and some navigation code. Associating simulated states of enemies with the received coordinates from the API provided an implementation challenge. This association is needed because the API provides only the location and not the current speed, which has to be derived from consecutive recordings. Without accurate enemy behavior simulation, this can be difficult, especially if there are several enemies in a close area.

b) A for Infinite Mario*: The *A* search algorithm* is a widely used best-first graph search algorithm that finds a path with the lowest cost between a pre-defined

start node and one out of possibly several goal-nodes[1]. A* uses a heuristic that estimates the remaining distance to the goal nodes in order to determine the sequence of nodes that are searched by the algorithm. It does so by adding the estimated remaining distance to the previous path cost from the start node to the current node. This heuristic should be admissible (not overestimate this distance) for optimality to be guaranteed. However, if only near-optimality is required, this constraint can be relaxed to speed up path-finding. With a heuristic that overestimates by x, the path will be at most x too long[2]. To implement an A* search for Infinite Mario, we have to define what a node and a neighbor of a node is, which nodes are goal nodes, and how the heuristic estimates the remaining distance. A node is defined by the entire world-state at a given moment in time, mainly characterized through Mario's position, speed, and state. Furthermore, (re)movable objects in the environment have to be taken into account, such as enemies, power-ups, coins and boxes. The only influence we have on this environment is interaction through actions that Mario can perform. These Mario actions consist of combinations of movements such as left, right, duck, jump and fire. Neighbors of a node are given by the world state after one (further) simulation step, applying an action that Mario

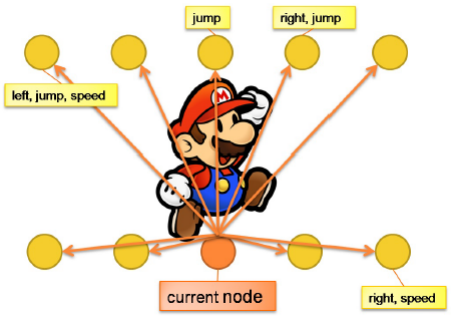


Fig. 3: Illustration of node generation in A* search. Each neighbour node in the search is given by a combination of possible input commands.

executes during this simulation step. See figures 3 and 4 for an illustration of a typical scenario, figure 5 for an in-game visualization. Note how backtracking takes place when an obstacle is discovered, and how the location of the subsequent neighbours is influenced by the current speed of Mario in figure 4. As only a small window of the world around Mario is visible in the API, the level structure outside of this window is unknown. Therefore it does not make sense to plan further ahead than the edge of the current window. Thus, a goal node for our planner is defined as any node where Mario is outside (or just before) the right border of the window of the known environment. In the standard implementation of A*, the heuristic estimates the remaining distance to the target. In Infinite Mario, one could just translate this as the horizontal distance to the right border of the window. However, this does

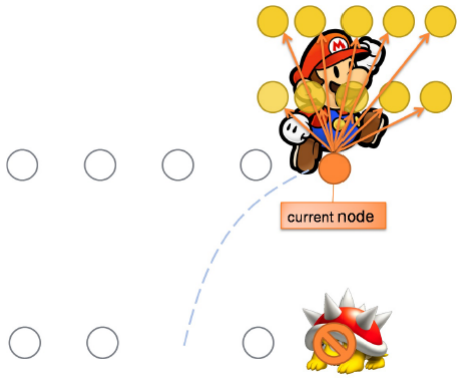


Fig. 4: Illustration of node placement in A* search in Infinite Mario. The best node in the previous simulation step (right, speed) is inaccessible because it contains an obstacle. Thus the next best node is chosen, which results in a jump over the obstacle. The search process continues by generating all possible neighbours. The speed of Mario at the current node distorts the positions of all following nodes.

not lead to a very accurate heuristic, as it ignores the current speed of Mario. To make sure the bot has an admissible (i.e., underestimating) heuristic, it would have to assume Mario runs with maximum speed towards the goal, which is often not

Fig. 5: Visualization of the future paths considered by the A* controller. Each read line shows a possible future trajectory for Mario, taking the dynamic nature of the world into account.

the case. Instead, a more accurate representation of the distance to the goal can be given by simulating the time required to reach the right border of the window. Here, the current speed of Mario is taken into account. The quickest solution to get to the

goal would then be to accelerate as fast as possible, and taking the required time as a heuristic. Similarly, the previous distance of a node to the starting node is simply the time it took to reach the current node.

c) Variable Optimisation: While the A* search algorithm is quite solid and guarantees optimality, these restrictions will likely lead to a non-optimal solution, so careful testing has to be undertaken to ensure that the search terminates. The first restriction used was to stop looking for solutions when the time-limit had been reached, and using the node with the best heuristic so far as a goal node. The linear level design created by the level generator in Infinite Mario favors this approach, as it does not feature dead ends that would require back-tracking of suboptimal paths.

The requirement that the heuristic has to be admissible has also been relaxed. A slightly overestimating heuristic increases the speed of the algorithm in expense of accuracy[2]. In detail, the estimation of the remaining distance can be multiplied with a factor w ¿= 1. Experimentation with different values for w, led to an optimal factor of w = 1.11. Another factor that has an effect on the processing time required by A* is the frequency of plan recalculation. As mentioned above, limited information requires a recalculation of the plan once new information becomes available, i.e., obstacles or enemies enter the window of visible information. Experimentation indicated that the best balance between planning ahead and restarting the planning process to incorporate new information is given when the plan is recreated every two game updates. Planning longer in advance occasionally led to reacting too late to previously unknown threats, resulting in a lost life or slowdown of Mario.

5 Results

Here is a link to a video showing the results of the A* controller implemented in Infinite Mario AI. `https://www.youtube.com/watch?v=CvI36-RgwlM`

6 Conclusions

A* performed very well once implemented, as A* is a commonly used algorithm for path-finding in many types of commercial computer games (e.g. RTS and MMORPG games), one could see this as a victory over classical AI over more fancy Computational Intelligence techniques which are rarely used in the games industry. However, one could also observe that the type of levels generated by the level generator are much less demanding and deceiving than those found in the real Super Mario Bros games and other similar games. All the levels could be cleared by constantly running right and jumping at the right moments, there were no hidden objects and passages, and in particular there were no dead ends that would require backtracking. The A*based agent consume considerably more processing time when in front of vertical walls, where most action sequences would not lead to the right

end of the screen, suggesting that A* would break down when faced with a dead end. While the sort of search in game-state space that the A* algorithm provides is likely to be an important component in any agent capable of playing arbitrary Mario levels, it will likely need to be complemented by some other mechanism for higher-level planning, and the architecture will probably benefit from tuning by e.g. evolutionary algorithms. Further, the playing style exhibited by the A*-based agent is nothing like that exhibited by a human.How to create a controller that can (learn to) play a platform game in a human-like style is industrially relevant problem which is not addressed by this work.

References

[1] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), pp. 100–107. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4082128.

[2] I. Millington and J. Funge. *Artificial Intelligence for Games*. Morgan Kaufmann. Taylor & Francis, 2009. ISBN: 9780123747310. URL: http://books.google.com.mx/books?id=1OJ8EhvuPXAC.

General Game Playing

Luis M Benitez

1 Introduction

General game playing is an application of AI that, in a way, takes us back to the days were AI searched for an all-encompassing program that could adapt itself to many situations, rather than todays focus on task-specific programs. The objective of GGP is to create a player a program- able to learn games without prior knowledge, and be able to play them skillfully.

One of the more prominent testbeds of GGP is the annual GGP Competition organized by Stanford University [6], for which a standardized language for game description, as well as a Game Manager, which acts as an enforcer of the game rules, have been created.

The language is GDL, which is very similar to Prolog in syntax. GDL is able to specify the initial and goal states, legal moves, state changes and player roles (e.g. X and O are roles for tic-tac-toe). Its description is found at [3]. The GM in turn is in charge of informing players of the game description; checking for the legality of moves; updating the global game state after each move and informing players of the new state; among other duties.

2 State of the art

Mandziuk et al. [4] describe the approaches of previous GGP competition winners. Cluneplayer was the first winner of the GGP in 2005. It analyzed the rules to extract features and group the into three aspects: expected payoff, control (mobility) and expected game termination. The features were then analyzed and combined based on their correlation to the game score they provided. Most other winners use Monte

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201, e-mail: a00994361@itesm.mx

Carlo (MC) simulations with UCT. The driving idea of MC is for the player to run as many random games as possible from the current state, and store the results for each path. UCT uses the information generated in the MC simulations to re-explore the paths with undiscovered paths or better results more often than others. UCT basically keeps a score for each node of the wins and exploration of each node. The rate of win/explorations is the criteria used to select the next node

Méhat and Cazenave [5] describe Ary, the winner of the 2009 GGP competition uses an UCT+T algorithm, and try out a method with a combination of nested MC and UCT called MAX. Nested MC chooses the next move at a level n based on the scores obtained on previous random playouts starting on that node. The UCT+T is UCT with transposition tables, the tables store visited positions so if they are arrived at from a different path, the player does not need to explore them again. The MAX approach splits the processing time between NMC and UTC+T.

Kobti and Sharma [2] use a set of hive players to find their strategies. When no knowledge is available, these players simply play random games, storing the better strategies in a knowledge base. Later iterations of the hive players use the stored strategies and some basic evolutionary mutations to continue searching. In their tests this player performs as well as a player that uses heuristics tailored to each game.

Swiechowski and Mandziuk [8] created a finalist player in the 2012 GGP competition by using a pool of playing strategies along with UCT. It is very similar to Ardy, but uses the strategy pool instead of a random MCTS (although one of their strategies is random search). The strategies are also evaluated dynamically, so the more successful strategies are picked more often for simulations, in an attempt to maximize the benefit of each simulation. Reported results are of an improved record of wins from 1 out of 9to 5 out of 9.

For a different approach, Sharma, Kobti and Goodwin [6] adapt ant search algorithms and reinforcement learning for their player. In addition they split the learning method by temporal levels, i.e. a set number of playing turns. This is to maximize the usefulness of features used in learning. The ants in their algorithm are the players, and the pheromone trails are limited by temporal level. The pheromone trail is the average score attained through m, and the move maximizing the reward is selected stochastically based on the ant (player) trail. Their test were run on a set of basic games, and other than in checkers, manage to obtain ¿65 % wins and kept losses to under 10% of games.

Sheng and Thuente [7] also begin their player with random plays. These are used to extract common features and action on the winners histories. Features are analyzed to see their usefulness for predicting winning and sub-goals are defined from features with highest usefulness. Based on these sub-goals, they create dynamic decision trees to calculate the fitness of each move. The trees are contextual to maximize the usefulness of the features. Their test is a comparison of players using random search, sub-goal learning, decision tree learning and contextual decision tree learning, obtaining bigger winning rate for all games for the contextual decision tree learning. The games vary in their branching factor from 7 to $>$1000.

3 Methodology

The chosen approach for our player is a simplified hive player similar to the one in[2], but each of the drones will be using a monte carlo simulation instead of random play. It is simplified, however, because it does not include a mutation component or a strategy database for each of the drones. We later compare this player with a random player and a simple monte carlo player.

Algorithm 7 Select Move Function

```
function SELECTMOVE(timeLimit)
    finishBy ← timeLimit − 1000
    moves ← getLegalMoves(currentState, currentRole)
    if moves.size > 1 then
        droneCount ← max((maxDrones/moves.size), 3)
        for i ← 1..moves.size do
            scoreList.Add(SPAWNPLAYER(finishBy, moves(i), droneCount))
        end for
        bestMove ← getIndexOfMaxValue(scoreList)
        return moves(bestMove)
    else
        return noop
    end if
end function
```

The idea behind this is to expand the exploration of the game tree without doing an exhaustive search. Each turn, the player iterates over each move, and requests from the state machine the following state that move would produce. This state is then handed over to a number of drones, which perform MC exploration starting from that state.

Algorithm 8 Spawn Drone Function

```
function SPAWNPLAYER(finishBy, move, droneCount)
    i, score ← 0
    while i <= droneCount do
        nextState ← stateMachine.nextState(currentState, move)
        droneSet.Add(DRONEPLAYER(finishBy, nextState)) ▷ droneSet is a pool of concurrent
threads
        i++
    end while
    for all drone ∈ droneSet do                ▷ Each drone terminates itself and returns a score.
        score+ = drone.score                   ▷ The set will manage the responses as they are received.
    end for
    averageScore ← (score/droneCount)
    return averageScore
end function
```

As the game server provides a timeout for a decision to be made, the drones are provided with a time in which they must finish. After each complete random exploration of a single path is completed, the score achieved is stored and a counter for the number of paths explored from that state is increased. After this, the drone checks if it still is within the allotted time. When the time runs out, an average of the obtained scores is returned (using the number of paths explored counter).

Algorithm 9 Drone Player

```
function DRONEPLAYER(finishBy, nextState)
    while currentTime < finishBy do
        singleScore ← stateMachine.exploreRandomPathFrom(nextState)
        score+ = singleScore
        attempts++
    end while
    averageScore ← (score/attempts)
    return averageScore
end function
```

Initially the maximum score obtained was returned, with the assumption that the player could obtain a better score by following the path where the highest possible score. The reasoning followed to change this and return an average score instead was that a branch could provide one very good score and a great amount of bad scores, so the chances of getting the best score, considering the opposing player would choose rationally instead of randomly, were lower than if a branch with a reasonable amount of good scores was followed. It is reasonable to assume the best choice for our player will be excluded by a rational opposing player, so we prefer to follow a path that offers many good options instead of one offering a few excellent options.

Regardin the parametrization of the player, originally, there was one available runtime parameters for the hive MC player, which was the number of drones created for each available move. The disadvantage of this approach was that the impact of a bigger number of available moves and the performance was much bigger, and, for cases where the amount of moves was small, the parallelism was not exploited sufficiently.

The parametrization was changed so that a maximum number of drones was provided, the player then allots the number of drones per available move. As the number of available moves approaches the maximum number of drones, the player will start to behave as a normal monte carlo player. As a way to limit the impact of this number being too low, a minimum number of drones per move was set, so as to try to obtain some advantage over normal MC players even for games with very large number of moves possible.

4 Experiments

The experiments will compare the designed player vs two other players : a montecarlo player and a random player. For each player, 10 matches for each of the following games are executed:

Game	Difficulty
Blocker	Green
Two Player Free-for-All Zero-sum	Yellow
Connect-four	Yellow-Orange
Tic-tac-toe	Orange
Tic-tac-chess	Red

The games were chosen based on the difficulty level as indicated in the stanford GGP games list site [1]. The intention is to cover the entire range of difficulty levels available. There is no quantitative measure of difficulty available, so the color scale used in the GGP site is adopted for our purposes. Each match are played individually, i.e. there are no concurrent matches running.

For the experiments, the server included in the GGP codebase was used to queue all the matches and obtain the resulting scores for each player. The server maintains a history of all matches and also presents a summary of the results. The time limit for each turn was set to 15 seconds and the initialization phase lasts 5 seconds. This has no impact for our setup since neither of the players has any initialization routines to be executed in this phase. The player was configured with 30 total drone players max, and a minimum of 3 drones per available move.

The comparison with the MC player is the primary test for the efficiency of this player, as this is a complex player, and MC is a primary component of many previous players in the competition. The test with the random player was originally thought of mor as a sanity test than an actual benchmark, as it was expected losing to it would imply losing against other more rational players.

5 Results

Following are the results of the experiments. The Score fields indicates the percentage of the accumulated rewards awarded to both players during all the matches played that was obtained by the hive player, e.g. if a total of 400 points were awarded during the 10 games to both players, and our player got 300 of those, then the score would be 75. The Wins/Ties/Loses indicate in how many matches, out of the total of 10 played, the hive player achieved that result, e.g. a 0 would indicate none of the matches had that outcome. Steps indicate the average duration , in turns, for each game.

Game	Wins	Ties	Loses	Score	Steps
Blocker	6	0	4	60	7.8
Two Player Free-for-All Zero-sum	4	1	5	45	30
Connect-four	9	0	1	90	8.7
Tic-Tac-Toe	4	1	5	45	7.1
Two-Player Tic-Tac-Chess	3	6	1	75	25.5

Table 1: Results for Games against MonteCarlo player

Game	Wins	Ties	Loses	Score	Steps
Blocker	1	0	9	10	8.9
Two Player Free-for-All Zero-sum	7	1	2	75	30
Connect-four	9	0	1	90	8.7
Tic-Tac-Toe	6	1	3	65	8.1
Two-Player Tic-Tac-Chess	1	9	0	100	27.7

Table 2: Results for Games against Random player

6 Conclusions

We noticed some surprising results, such as the Random player being markedly better in the Blocker game, and moderately better in the Tic-Tac-Chess, than the MC player, which we expected would obtain better results. It is also observed that for Tic-Tac-Toe the performance is generally poor. This could be caused by the policy of following a path with several moderately good options over one with few very good options. In a game like tic-tac-toe, this strategy could cause the player to quickly lock itself out of winning possibilities. This sounds reasonable when looking at the amount of ties obtained in the Tic-Tac-Chess game.

One of the first points of improvement considered after analysing the results is to take advantage of the initial setup time provided by the game manager. This initial exploration could provide very valuable insights about the paths available. The second point is to include a persistent game tree for the match, so the information obtained through each exploration can be reused in later explorations to avoid paths previously found to offer poor scores and focus on those that are either unexplored or with good scores. This is essentially adding a UCT algorithm on top of the normal MC explorations.

References

[1] GGP.org. *Tiltyard Games*. Nov. 2013. URL: http://www.ggp.org/view/tiltyard/games/.

[2] Z. Kobti and S. Sharma. "A Multi-Agent Architecture for Game Playing". In: *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. 2007, pp. 276–281. DOI: 10.1109/CIG.2007.368109.

[3] Nathaniel Love et al. *General Game Playing: Game Description Language Specification*. 2008.

[4] Jacek Mandziuk, Yew Soon Ong, and Karol Waledzik. "Multi-game playing: A challenge for computational intelligence". In: *Computational Intelligence for Human-like Intelligence (CIHLI), 2013 IEEE Symposium on*. 2013, pp. 17–24. DOI: 10.1109/CIHLI.2013.6613260.

[5] J. Mehat and T. Cazenave. "Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing". In: *Computational Intelligence and AI in Games, IEEE Transactions on* 2.4 (2010), pp. 271–277. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2010.2088123.

[6] S. Sharma, Z. Kobti, and S.D. Goodwin. "General Game Playing: An Overview and Open Problems". In: *Computing, Engineering and Information, 2009. ICC '09. International Conference on*. 2009, pp. 257–260. DOI: 10.1109/ICC.2009.50.

[7] Xinxin Sheng and D. Thuente. "Contextual Decision Making in General Game Playing". In: *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*. 2011, pp. 679–684. DOI: 10.1109/ICTAI.2011.108.

[8] M. Swiechowski and J. Mandziuk. "Self-Adaptation of Playing Strategies in General Game Playing". In: *Computational Intelligence and AI in Games, IEEE Transactions on* PP.99 (2013), pp. 1–1. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2013.2275163.

Training a Tetris player agent with a genetic algorithm

Guillermo Antonio Palomino Sosa

1 Tetris player

1.1 Introduction

Tetris is a video game created by Aleksi Pzhitnov in 1984. It is available in almost any computer system. The game is played for one player that has to accommodate the Tetris pieces and form horizontal lines. This paper describes the components to create a AI player that plays the Tetris game. An agent with heuristic rules will be develop and trained with a genetic algorithm.

1.2 Tetris rules

There are 7 original Tetris pieces showed in figure. The game starts with an empty board and one by one a random figure appears on the top of the board. The figure that appears starts to descend to the bottom of the board and to pile up over the pieces on the board. Each time that the player forms an horizontal line with the pieces, this line disappears and the player earn some points. The player can rotates the pieces that are descending but it cant move the piled up pieces on the board. The game ends when there is not enough spaces for a new piece to appear on the top of the board.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

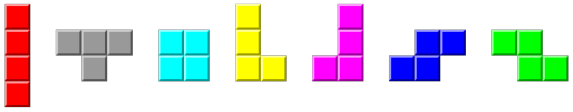


Fig. 1: Tetris pieces

1.3 Heuristic rules

The strategy is divided into penalize and reward the moves. The AI player will take the decision trying to get the biggest reward

Penalize.

Number of holes: Penalizes for each hole on the board after the piece placement.
Height: Penalizes for the increasing of the number of rows with blocks on it.
Block contact: Reward for the number of blocks that have contact with other blocks
Wall contact: Reward for the number of blocks that are in contact with the wall
Floor contact: Reward for the number of blocks that are on the floor
Scoreline: Reward if after the placement of the piece a new line or more are complete.

The Figure 2 shows examples of the heuristic rules.

2 Implementation description

An existing Tetris implementation in Java was taken from the user PSNBTech of Github[4]. The game was developed to be played by a human. A control and monitoring layer was added to enable the Tetris player agent to control the game. Figure 3 shows the software architecture used.

2.0.1 Control layer

The control layer exposes a control functions on the Tetris game that were designed to be accessed by a human player through the keyboard. Functions has rotate, move left, move right are available. The control layer also provides monitoring capabilities. The state of the Tetris board can be accessed.

2.0.2 Tetris agent

The Tetris agent is in charge of playing the game instead of a human being. It knows the Tetris rules and is provided of the heuristic rules to take decisions on the best

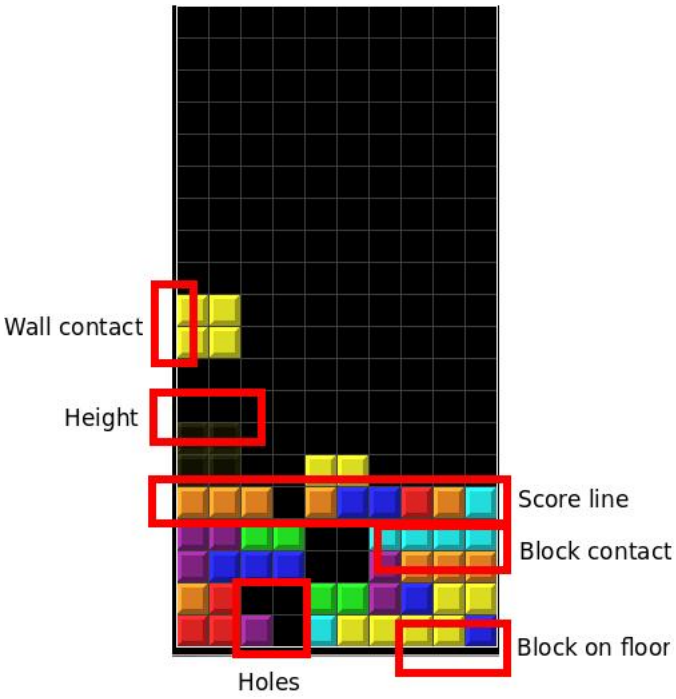


Fig. 2: Heuristic examples

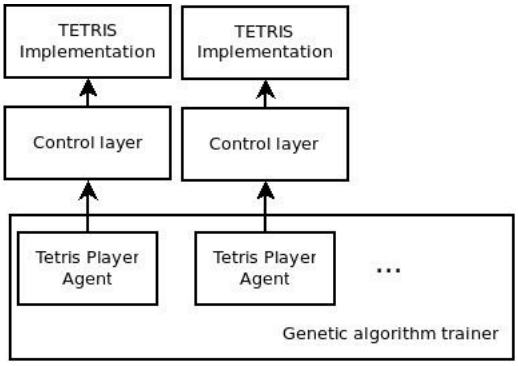


Fig. 3: Software stack

move[3]. To chose the best move, the Tetris agent evaluates all the available positions where the new piece can be placed in all the possible rotations as shown in figure 4.

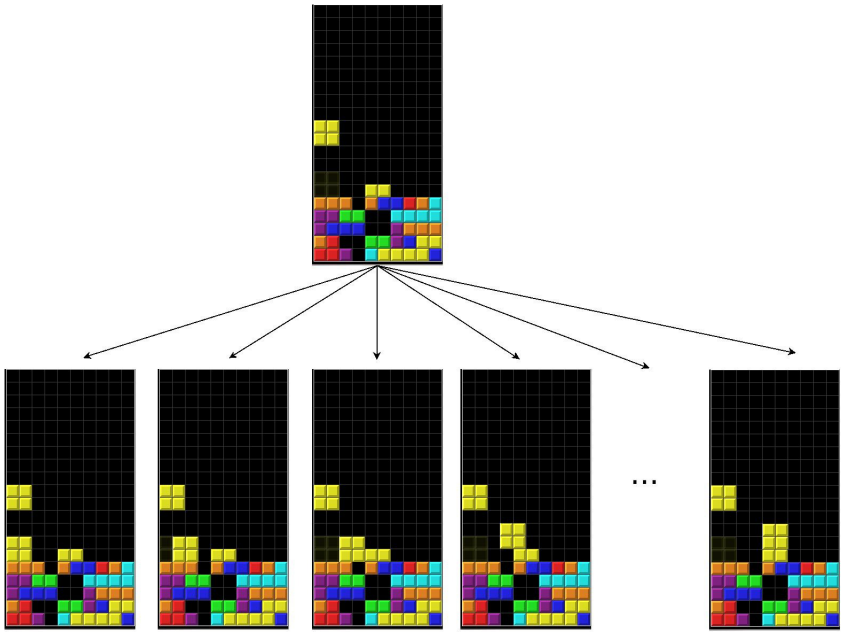


Fig. 4: Neighbor generator

The decision on the next move is taken as is indicated on algorithm 10

Algorithm 10 Move Decision

```
R = 0
for R < 4

    R = R + 1
    Score = 0
    BestPosition = null
    for    x < numberRows do
        for do    y < numberColumns
            set    piece in position x,y with rotation R
            if (position    is valid)
                newScore =    CalculateScore(x,y,R)
                if    (newScore > Score)
                    Score = newScore
                    BestPosition = position(x,y,R)
                endif
            endif
            y = y + 1
        end for
        x = x + 1
    end for

end loop
```

The score is calculated as indicated in algorithm 11

Algorithm 11 Score calculation

Input: position x,y and rotation R
Output: *Score*
$Score = 0$
$Score = Score$+number of generated holes $*$heuristic value of holes
$Score = Score$+height of the board $*$heuristic value of height
$Score = Score$+number of blocks touching the wall $*$heuristic value of wall touching
$Score = Score$+number of generated lines $*$heuristic value of line generation
$Score = Score$+number of blocks touching the floor $*$heuristic value of floor touching
$Score = Score$+number of blocks touching each other $*$heuristic value of block to block touching

3 Genetic algorithm

The following elements of the genetic algorithm have been identified:[5] [1]

Chromosome:

its the set of float values used on the score calculation as multiplier (algorithm 11).

Fitness function:

Is the score given by the Tetris game when the game is over. The score is calculated based on the number of complete lines done on the game.

Population:

Is a set of Tetris player agents loaded with certain chromosome.

The genetic algorithm showed on Algorithm 12 is executed until the MAX_GEN is reached. MAX_GEN is used as limit for the program. The algorithm generates a random generation for the first iteration and the following generations will be composed of the chromosomes from the best previous generations.

Algorithm 12 Genetic algorithm

```
GenCounter = 0
for GenCounter <MAX_GEN

    if (GenCounter = 0)
        set random generation
    else
        mix generations and create     players with generated chromosomes
    endif
    Run next generation
    GenCounter = GenCounter + 1

endfor
```

The Genetic algorithm uses a mix generations function described in Algorithm 13[2]. This function uses a tournament selection. a pair of scores are selected and worst score will be deleted. A pair of best chromosomes will be selected and their attributes will be randomly mixed up. There will be a 30% probability of mutation during the attribute mixing.

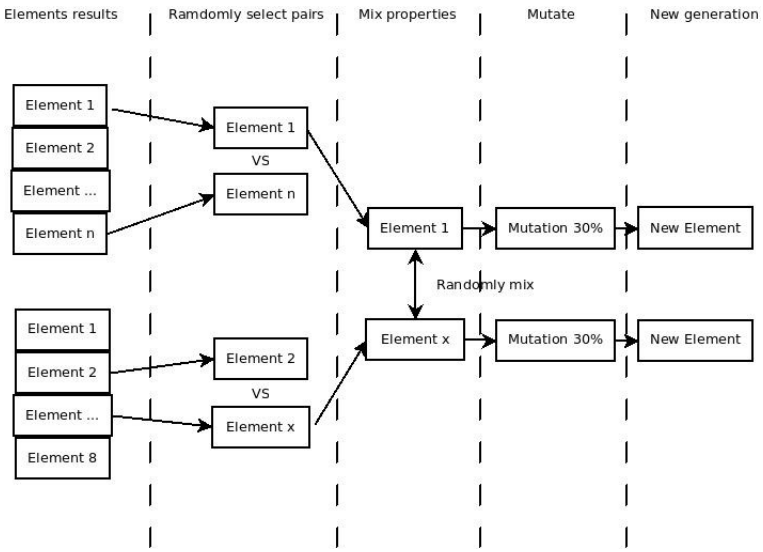


Fig. 5: Mixing

Algorithm 13 Mix generations function

```
for each     chromosome of the population

    select     a pair of chromosomes
    keep     only best chromosome

endfor
for each     best chromosome

    select     a pair of best chromosomes
    for each     attribute of the chromosome
        if (random() > 0.5)
            mix     attributes between the pair of chromosomes selected
        endif
        if (random()> 0.3)
            set     random attribute to the chromosomes
        endif
    endfor
    set     new mixed up chromosomes on the list for to be assigned on the next generation

endfor
```

4 Results

The following parameters were used to execute the test:

Number of generations: 80

Population number: 8

The log generated is composed by the following elements:

Generation Number
ID of element N, Score obtained, Heuristic values
Genealogical tree of the element
ID of element N+1, Score obtained, Heuristic values
Genealogical tree of the element
...

Nomenclature used:

R: Score result
BW: Heuristic value for blocks touching the wall
HT: Heuristic value for the height
CL: Heuristic value for the lines completed after the position of the piece
HO: Heuristic values for the holes generated after the position of the piece
CB: Heuristic value for the contact between blocks
BF: Heuristic value for the blocks touching the floor of the board.

On the first generation almost all elements obtained score results of 0:

```
Generation 0
ID: 0 ,R: 0.0 ,BW: 4.020374 ,HT: -18.879704 ,CL: -7.4799714 ,HO: 4.7368917 ,HO: 2.2391336 ,BF: 8.74207
ID: 1 ,R: 100.0 ,BW: -15.100175 ,HT: -9.182279 ,CL: 19.410019 ,HO: 4.9705534 ,HO: 10.952796 ,BF: -8.677019
ID: 2 ,R: 0.0 ,BW: 17.92854 ,HT: 7.8272834 ,CL: -7.3891835 ,HO: -2.747564 ,HO: -4.3507533 ,BF: 19.64611
ID: 3 ,R: 0.0 ,BW: 11.773256 ,HT: -16.155663 ,CL: 9.697473 ,HO: 3.221829 ,HO: 0.05047871 ,BF: -2.0074315
ID: 4 ,R: 200.0 ,BW: -9.979408 ,HT: -17.39301 ,CL: 11.045591 ,HO: -0.6888518 ,HO: 14.208897 ,BF: 13.334054
ID: 5 ,R: 0.0 ,BW: 1.2547315 ,HT: -10.643291 ,CL: -12.163822 ,HO: 0.072104834 ,HO: 4.8975606 ,BF: -12.006707
ID: 6 ,R: 0.0 ,BW: 19.16935 ,HT: -12.744434 ,CL: -8.995025 ,HO: 9.116133 ,HO: -0.16312414 ,BF: 2.1119423
ID: 7 ,R: 0.0 ,BW: 18.23838 ,HT: 2.9070778 ,CL: 0.30952767 ,HO: 8.844369 ,HO: 9.415101 ,BF: 9.215477
```

On generation 11 element 51 obtained 1200 points:

```
Generation 11 ID: 47 ,R: 0.0 ,BW: -9.979408 ,HT: -7.757071 ,CL: -3.354696 ,HO: 4.9705534 ,HO: -1.318587 ,BF:
      -12.587265
(4,17)(35,22)(38,37)
ID: 37 ,R: 100.0 ,BW: -9.979408 ,HT: -2.418907 ,CL: 19.410019 ,HO: 4.9705534 ,HO: 14.208897 ,BF: -12.587265
(4,17)(35,22)
ID: 48 ,R: 0.0 ,BW: -9.979408 ,HT: -18.391216 ,CL: -3.354696 ,HO: 4.9705534 ,HO: -1.318587 ,BF: -12.587265
(4,17)(35,22)(38,37)(47,37)
ID: 49 ,R: 0.0 ,BW: -9.979408 ,HT: 19.03542 ,CL: 19.410019 ,HO: 4.9705534 ,HO: 14.208897 ,BF: -12.587265
(4,17)(35,22)(47,37)
ID: 38 ,R: 200.0 ,BW: -9.979408 ,HT: -17.39301 ,CL: -3.354696 ,HO: 8.27542 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)
ID: 46 ,R: 0.0 ,BW: 4.7526865 ,HT: 9.806016 ,CL: 11.08584 ,HO: 8.27542 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)
ID: 50 ,R: 300.0 ,BW: -9.979408 ,HT: -17.39301 ,CL: -3.354696 ,HO: 8.27542 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,46)
*ID: 51 ,R: 1200.0 ,BW: 4.7526865 ,HT: 9.806016 ,CL: 11.08584 ,HO: -17.843576 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)
```

On the generation 12, the descendent of element 51 and 37 (53) obtained the best result. The element didn't has any mutation

```
Generation 12
ID: 37 ,R: 400.0 ,BW: -9.979408 ,HT: -2.418907 ,CL: 19.410019 ,HO: 4.9705534 ,HO: 14.208897 ,BF: -12.587265
(4,17)(35,22)
ID: 51 ,R: 1500.0 ,BW: 4.7526865 ,HT: 9.806016 ,CL: 11.08584 ,HO: -17.843576 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)
ID: 52 ,R: 0.0 ,BW: -9.979408 ,HT: 9.806016 ,CL: 11.08584 ,HO: -17.843576 ,HO: 14.208897 ,BF: -12.587265
(4,17)(35,22)(37,51)
ID: 53 ,R: 2200.0 ,BW: 4.7526865 ,HT: -2.418907 ,CL: 19.410019 ,HO: 4.9705534 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(37,51)
ID: 50 ,R: 200.0 ,BW: -9.979408 ,HT: -17.39301 ,CL: -3.354696 ,HO: 8.27542 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,46)
ID: 38 ,R: 300.0 ,BW: -9.979408 ,HT: -17.39301 ,CL: -3.354696 ,HO: 8.27542 ,HO: 14.208897 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)
ID: 54 ,R: 100.0 ,BW: 17.187702 ,HT: -17.39301 ,CL: -3.354696 ,HO: 8.27542 ,HO: -7.560815 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(38,46)(50,38)
ID: 55 ,R: 0.0 ,BW: -9.979408 ,HT: -17.39301 ,CL: -3.354696 ,HO: 8.27542 ,HO: -18.705523 ,BF: 13.334054
(8,1)(1,15)(4,17)(25,23)(29,32)(50,38)
```

By generation 80 all elements are obtaining excellent Scores. 327 elements have been generated on generation 80. As the players become better, the time for the to lose increase.

```
Generation 80
ID: 320 ,R: 14300.0 ,BW: 11.139992 ,HT: -19.263601 ,CL: 12.7325115 ,HO: -7.925561 ,HO: 14.208897 ,BF:
      -0.41197464
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)(166,154)
(193,170)(201,219)(227,220)(227,243)(265,251)(279,278)(280,287)(292,307)(310,300)(314,310)
ID: 311 ,R: 352700.0 ,BW: 11.139992 ,HT: -2.418907 ,CL: 12.7325115 ,HO: -7.6635175 ,HO: 14.208897 ,BF:
      10.286173
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)
(166,154)(193,170)(201,219)(227,220)(227,243)(292,251)(292,307)
ID: 324 ,R: 361400.0 ,BW: 11.139992 ,HT: -2.418907 ,CL: 12.7325115 ,HO: -7.6635175 ,HO: 14.208897 ,BF:
      10.286173
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)(166,154)
(193,170)(201,219)(227,220)(227,243)(265,251)(279,278)(280,287)(292,307)(310,300)(314,310)(320,311)
ID: 325 ,R: 15100.0 ,BW: 11.139992 ,HT: -19.263601 ,CL: 12.7325115 ,HO: -7.925561 ,HO: 14.208897 ,BF:
      -0.41197464
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)(72,67)
(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)(166,154)(193,170)
(201,219)(227,220)(227,243)(292,251)(292,307)(320,311)
ID: 322 ,R: 1900.0 ,BW: 10.180863 ,HT: -2.418907 ,CL: -3.0309837 ,HO: -7.6635175 ,HO: 14.208897 ,BF: 10.286173
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)
(166,154)(193,170)(201,219)(227,220)(227,243)(265,251)(279,278)(280,287)(292,307)(310,300)(314,311)(316,311)
ID: 316 ,R: 353700.0 ,BW: 10.180863 ,HT: -2.418907 ,CL: 19.209457 ,HO: -7.6635175 ,HO: 14.208897 ,BF:
      10.286173
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)
(166,154)(193,170)(201,219)(227,220)(227,243)(265,251)(279,278)(280,287)(292,307)(310,300)(314,311)
ID: 326 ,R: 12000.0 ,BW: 10.180863 ,HT: 7.32755 ,CL: 19.209457 ,HO: -7.6635175 ,HO: 14.208897 ,BF: 10.286173
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)
(166,154)(193,170)(201,219)(227,220)(227,243)(265,251)(279,278)(280,287)(292,307)(310,300)(314,311)(316,311)(322,316)
ID: 327 ,R: 65100.0 ,BW: 10.180863 ,HT: -2.418907 ,CL: -3.0309837 ,HO: -7.6635175 ,HO: 14.208897 ,BF:
      10.286173
(8,1)(1,15)(4,17)(25,23)(29,32)(38,37)(38,46)(51,53)(53,58)(61,60)(61,64)(69,67)
(72,67)(80,76)(81,89)(95,93)(94,99)(103,94)(94,106)(115,117)(130,135)(154,157)
(166,154)(193,170)(201,219)(227,220)(227,243)(265,251)(279,278)(280,287)(292,307)(310,300)(314,311)(322,316)
```

5 Conclusion

An approach of calculate all the possible states of a Tetris game is impossible. The use of a agent is a better approach to solve the problem. Many problems require tuning parameters to perform properly. Tetris player is a great example of a problem that need to be tuned. The algorithm determines what parameters are more valuable and if those parameters should reward or penalize the score. The mutation rate prevents the algorithm from stuck.

References

[1] Amine Boumaza. "Designing artificial tetris players with evolution strategies and racing". In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. GECCO '11. Dublin, Ireland: ACM, 2011, pp. 117–118. ISBN: 978-1-4503-0690-4. DOI: 10.1145/2001858.2001925. URL: http://doi.acm.org/10.1145/2001858.2001925.

[2] Amine Boumaza. "Reducing the learning time of tetris in evolution strategies". In: *Proceedings of the 10th international conference on Artificial Evolution*. EA'11. Angers, France: Springer-Verlag, 2012, pp. 193–204. ISBN: 978-3-642-35532-5. DOI: 10.1007/978-3-642-35533-2_17. URL: http://dx.doi.org/10.1007/978-3-642-35533-2_17.

[3] Hendrik Jan Hoogeboom and Walter A. Kosters. "Tetris and decidability". In: *Inf. Process. Lett.* 89.6 (Mar. 2004), pp. 267–272. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2003.12.006. URL: http://dx.doi.org/10.1016/j.ipl.2003.12.006.

[4] PSNBTech. *PSNBTech / Tetris*. Nov. 2013. URL: https://github.com/PSNBTech/Tetris.

[5] Howard Yeend. *Genetic Algorithm For Hello World*. Dec. 2010. URL: http://www.puremango.co.uk/2010/12/genetic-algorithm-for-hello-world.

Chapter 4
Other applications

A Comparison of Global Index of Stock Markets through Neural Networks

Francisco Aguirre

1 Introduction

Prediction of stock market returns is an important issue in finance. Artificial neural networks have been used in stock market prediction during the last decade. Studies were performed for the prediction of stock index values as well as daily direction of change in the index. In some applications it has been specified that artificial neural networks have limitations for learning the data patterns or that they may perform inconsistently and unpredictable because of the complex financial data used.

The ability of neural networks to closely approximate unknown functions to any degree of desired accuracy has generated considerable demand for neural network research in Business. The attractiveness of neural network research stems from researchers need to approximate models within the business environment without having a priori knowledge about the true underlying function. However, despite all advantages cited for artificial neural networks, their performance for some real time series is not satisfactory. Improving forecasting especially time series forecasting accuracy is an important yet often difficult task facing forecasters. Both theoretical and empirical findings have indicated that integration of different models can be an effective way of improving upon their predictive performance, especially when the models in the ensemble are quite different. Artificial Neural Networks are flexible computing frameworks and universal approximators that can be applied to a wide range of time series forecasting problems with a high degree of accuracy.

Time series S is represented as an ordered sequence:

$$S = (x_1, x_2, ..., x_n)$$

where x is the observed value a time .

For certain types of problems, such as learning to interpret complex real-world sensor data, artificial neural networks are among the most effective learning methods

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

currently know. During the last decade they have been widely applied to the domain of financial time series prediction and their importance in this field is growing.

Investing in stocks is one of the easy money making in short duration. Even though with high risk, every investor wants to make profit by investing a good amount of money in share market. As investors are investing more and more money in the market, they are very much concerned to know the future trends of an assortment of stocks available in the market. The major part of the trends in the market is to know when to buy, hold or sell the stocks. Thus, many models have been illustrated to provide the investors a correct trend of predictions. Traditionally, the various statistical models have been introduced for forecasting the stock index. Traditional statistical models cannot be used to track the complexity of the market behavior and economic theory. Investigations done by the researchers on traditional methods suggested that the prediction of financial time series fails because the stock markets are highly non-linear (change with time) and non-stationary (dynamic). Search methods are still on to address these shortcomings. Market analysts and academicians in finance, economics and business have begun to investigate the tools of computational intelligence, including nonlinear methods that incorporate machine learning and other advanced computational technologies.

In this paper my aim will be to analyze neural networks method used in artificial intelligence to forecast the future trends of the main Stock Market Index in Americas, Europe and Asia.

The main index we will use are:

AMERICAS Nasdaq, Bovespa, IPC EUROPE CAC40, DAX ASIA Nikkei

In the past several decades, as a result of increased globalization and technology explosions, world financial markets are more and more interrelated and integrated. The development has enhanced the transfer of information flows from one market to another across national boundaries. In response to these changes, there comes a diverse amount of literature on the topic of information transmission in world stock market. Many empirical studies try to find the information transmission mechanism.

Prediction of stock market returns is an important issue in finance. Nowadays artificial neural networks (ANNs) have been popularly applied to finance problems such as stock exchange index prediction, bankruptcy prediction and corporate bond classification. An ANN model is a computer model whose architecture essentially mimics the learning capability of the human brain. The processing elements of an ANN resemble the biological structure of neurons and the internal operation of a human brain. Many simple interconnected linear or nonlinear computational elements are operating in parallel processing at multiple layers. In some applications it has been specified that ANNs have limitations for learning the data patterns. They may perform inconsistently and unpredictable because of the complex financial data used. Sometimes data is so voluminous that learning patterns may not work. Continuous and large volume of data needs to be checked for redundancy and the data size should be decreased for the algorithm to work in a shorter time and give more generalized solutions.

1.1 Artificial Neural Network Approach

Machine learning approach is appealing for artificial intelligence since it is based on the principle of learning from training and experience. Connectionist models, such as ANNs, are well suited for machine learning where connection weights are adjusted to improve the performance of a network. An ANN is a network of nodes connected with directed arcs each with a numerical weight, w_{ij}, specifying the strength of the connection (Figure-1). These weights indicate the influence of previous node, u_j, on the next node, u_i, where positive weights represent reinforcement; negative weights represent inhibition [7].

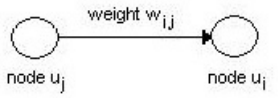


Fig. 1: Connection weight between nodes.

Generally the initial connection weights are randomly selected. Input layer is composed of a set of inputs that feed input patterns to the network. Following the input layer there will be at least one or more intermediate layers, often called hidden layers. Hidden layers will then be followed by an output layer, where the results can be achieved (Figure-2). In feedforward networks all connections are unidirectional.

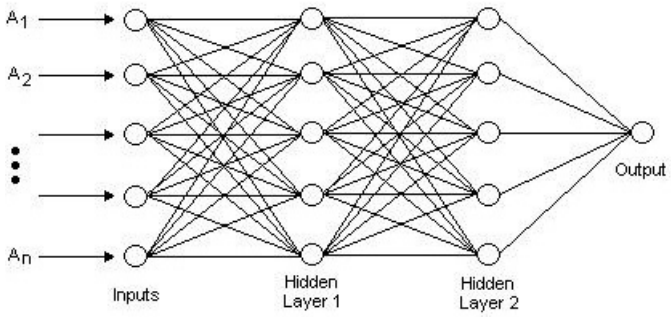


Fig. 2: 2-hidden layers network with n inputs and 1 output.

1.2 Analytical Methods

Before the age of computers, people traded stocks and commodities primarily on intuition. As the level of investing and trading grew, people searched for tools and methods that would increase their gains while minimizing their risk. Statistics, technical analysis, fundamental analysis, and linear regression are all used to attempt to predict and benefit from the markets direction. None of these techniques has proven to be the consistently correct prediction tool that is desired, and many analysts argue about the usefulness of many of the approaches. However, these methods are presented as they are commonly used in practice and represent a base-level standard for which neural networks should outperform. Also, many of these techniques are used to preprocess raw data inputs, and their results are fed into neural networks as input.

1.3 Technical Analysis

The idea behind technical analysis is that share prices move in trends dictated by the constantly changing attitudes of investors in response to different forces. Using price, volume, and open interest statistics, the technical analyst uses charts to predict future stock movements. Technical analysis rests on the assumption that history repeats itself and that future market direction can be determined by examining past prices. Thus, technical analysis is controversial and contradicts the Efficient Market Hypothesis. However, it is used by approximately 90because it is highly subjective. Different individuals can interpret charts in different manners. Price charts are used to detect trends. Trends are assumed to be based on supply and demand issues which often have cyclical or noticeable patterns. There are a variety of technical indicators derived from chart analysis which can be formalized into trading rules or used as inputs to neural networks. Some technical indicator categories include filter indicators, momentum indicators, trend line analysis, cycle theory, volume indicators, wave analysis, and pattern analysis. Indicators may provide short or long term information, help identify trends or cycles in the market, or indicate the strength of the stock price using support and resistance levels. An example of a technical indicator is the moving average. The moving average averages stock prices over a given length of time allowing trends to be more visible. Several trading rules have been developed which pertain to the moving average. For example, ”when a closing price moves above a moving average a buy signal is generated”. Unfortunately, these indicators often give false signals and lag the market. That is, since a moving average is a past estimate, a technical trader often misses a lot of the potential in the stock movement before the appropriate trading signal is generated. Thus, although technical analysis may yield insights into the market, its highly subjective nature and inherent time delay does not make it ideal for the fast, dynamic trading markets of today.

1.4 Fundamental Analysis

Fundamental analysis involves the in-depth analysis of a companys performance and profitability to determine its share price. By studying the overall economic conditions, the companys competition, and other factors, it is possible to determine expected returns and the intrinsic value of shares. This type of analysis assumes that a shares current (and future) price depends on its intrinsic value and anticipated return on investment. As new information is released pertaining to the companys status, the expected return on the companys shares will change, which affects the stock price. The advantages of fundamental analysis are its systematic approach and its ability to predict changes before they show up on the charts. Companies are compared with one another, and their growth prospects are related to the current economic environment. This allows the investor to become more familiar with the company. Unfortunately, it becomes harder to formalize all this knowledge for purposes of automation (with a neural network for example), and interpretation of this knowledge may be subjective. Also, it is hard to time the market using fundamental analysis. Although the outstanding information may warrant stock movement, the actual movement may be delayed due to unknown factors or until the rest of the market interprets the information in the same way. However, fundamental analysis is a superior method for long-term stability and growth. Basically, fundamental analysis assumes investors are 90examining their investments in detail, whereas technical analysis assumes investors are 90reacting to changes in the market environment in predictable ways.

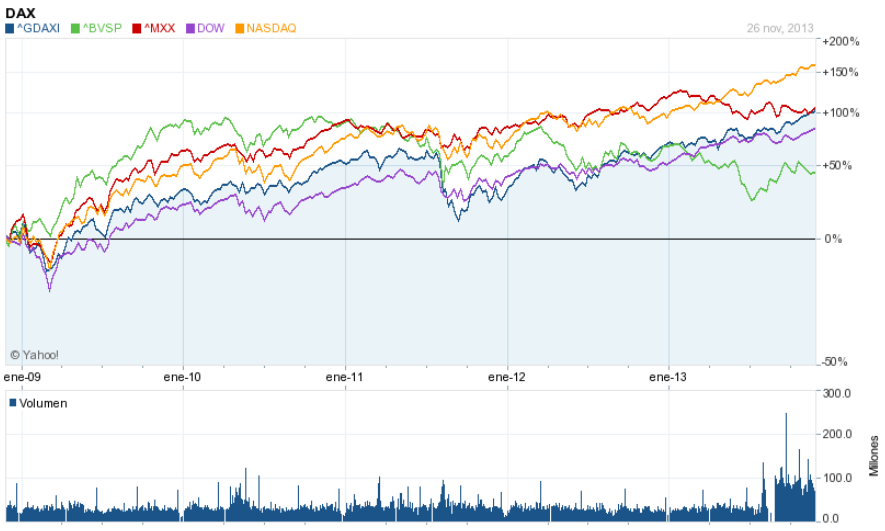


Fig. 3: Global Stock Markets indexes

2 Methodology

2.1 System Model

In this study the following input variables were considered to ultimately affect the stock exchange market index value:

BVSP Bovespa FCHI CAC40 GDAXI DAX SHSZ300 Shangahai Shenzhen CSI 300 Index NKY N225 Nikkei 225 CCMP Nasdaq Composite Index

3 Experiments

Several studies relating to ANN and statistical models have been conducted in the literature. Traditional forecasting methods are limited in their effectiveness as they make assumptions about the distribution of the underlying data, and often fail to recognize the interrelatedness of variables. Both linear and nonlinear models were used to predict stock returns who emphasize the Nonlinear Model proving to be more effective. Such studies prove that the nonlinear model presents more consistent results for stock exchange market. For this reason, ANN applications have been widely used in a variety of areas in financial markets. Reference confirmed that ANN was used for the solution of numerous financial problems. References emphasized that ANN could be used in the prediction of financial markets, in particular, the prediction of stock market indexes which are considered to be a barometer of the markets in many countries. Empirical evidence suggests that although these models appear to be capable of explaining the movements of major exchange rates in the long run and in economies experiencing hyperinflation, their performance is poor when it comes to the short run and out-of-sample forecasting. Conventional time series models forecasting on global approximation models, employing techniques such as linear and non-linear regression, polynomial fitting and artificial neural networks. Such models are better suited to problems with stationary dynamics. In and the application of unsupervised clusters for the segmentation of the input space, and feed forward neural networks (FNNs) acting as local predictors for each identified cluster, was proposed. Neural network researchers and developers using the generalized method for determining the minimum necessary training set size will be able to implement neural networks with the highest forecasting performance at the least cost.

3.0.1 Conclusions

- The prediction models based on ANNs were more accurate when volumne of that increased. Needs more data, possibly more types of data.

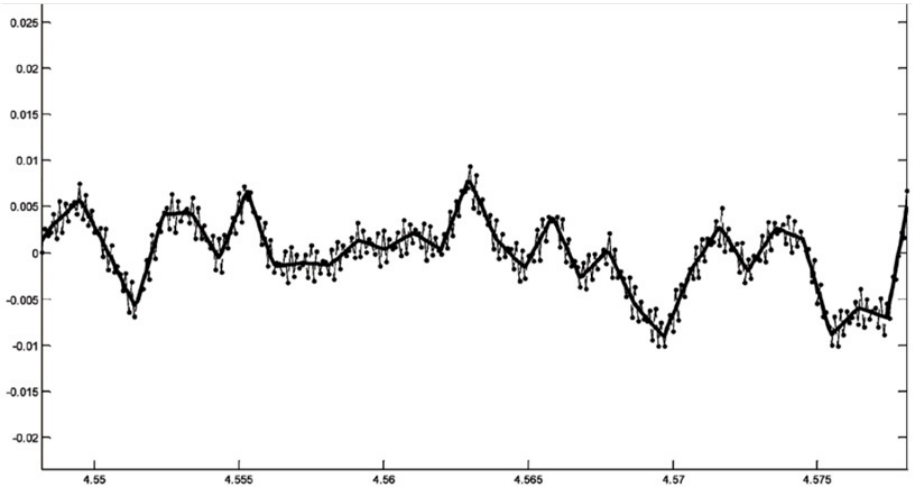


Fig. 4: IPC and ANN

- Neural Networks model shows promise, but needs improvement before becoming an effective aid.
- No human or computer can perfectly predict the volatile stock market
- Under normal conditions, in most cases, a good neural network will outperform most other current stock market predictors and be a very worthwhile, and potentially profitable aid to investors
- Should be used as an aid only

References

[1] Dirk Emma Baestaens and Willem Max van den Bergh. Tracking the Amsterdam stock index using neural networks. In Neural Networks in the Capital Markets, chapter 10, pages 149162. John Wiley and Sons, 1995.

[2] K. Bergerson and D. Wunsch. A commodity trading model based on a neural network-expert system hybrid. In Neural Networks in Finance and Investing, chapter 23, pages 403410. Probus Publishing Company, 1993.

[3] Robert J. Van Eyden. The Application of Neural Networks in the Forecasting of Share Prices. Finance and Technology Publishing, 1996.

[4] K. Kamijo and T. Tanigawa. Stock price pattern recognition: A recurrent neural network approach. In Neural Networks in Finance and Investing, chapter 21, pages 357370. Probus Publishing Company,1993

[5] Lawrence R., Using Neural Networks to Forecast Stock Market Prices, Department of Computer Science University of Manitoba, December 12, 1997.

[6] Kimoto T., K. Asakawa, M. Yoda, M. Takeoka, Stock market prediction system with modular neural network, Proceedings of the International Joint Conference on Neural Networks, 1-6, 1990.

[7] Mizuno H., M. Kosaka, H. Yajima, N. Komoda, Application of Neural Network to Technical Analysis of Stock Market Prediction, Studies inInformatic and Control, Vol.7, No.3, pp.111-120, 1998.

Genetic Algorithms: Timetabling problem

José Carlos Martínez and Jorge Antonio Origel

1 Introduction

The artificial intelligence (AI) has different applications. The different studies have developed different ways to handle real problems that we deal in a daily basis. This area is very complex for different scenarios, we can find different applications like in medicine, robotics or even our smart phones. This essay will show the application for a timetable problem.

The timetable problem can be applied in different ways, one could be in the industry, to have the different schedules for each employee, specially when you have restrictions of time, headcounts or tools. Other application for this problem is in the educational area. You can define the schedules for all the different classes and students, even if you have different restrictions like the number of available classrooms, or the number of available teachers for each subject, timings, etc.

The problem we are going to solve is in the educational area with the application of the genetic algorithms, understanding why using genetic algorithms, when should them be used and how is the implementation of them. We know that we can find different solutions for the same problem, but the objective is to maximize the use of the resources to bring up the best solution for the restrictions defined.

2 Genetic Algorithms

The Genetic Algorithms (GA) are inspired in the nature, specifically in the Darwin's evolution theory and the natural selection. Darwin said that the best individual will

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

survive accordingly to their strength and the capability to adapt themselves to the environment within an evolution process. GA try to simulate this idea. First, we begin with an initial population, then we start generating a new population based in the initial one by selecting two of the best individuals, get a crossover between those two individuals, and finally perform a mutation to that child depending on a probability. This process will continue until the first generation is complete and then we start the process again with the first generation of children as parents.

The idea sounds fuzzy at the beginning, but meanwhile we create new generations, we are creating new solutions, since we are choosing the best solutions for the crossover, each iteration we will be closer to the solution that fits all the requirements for the problem. Russell and Norvig talk about different the characteristics of these algorithms [3]:

a. Initial Population
Begin with a set of K randomly generated states.

b. Fitness Function
Each state is rated by the objective function or the fitness function, it should return a higher value for better states.

c. Selection
Pairs are selected at random for reproduction, in accordance with the probabilities in the fitness function.

d. Crossover
For each pair to be mated, a crossover point is chosen randomly from the positions in the string(When two parent states are quite difference, the crossover operation can product a state that is a long way from either parent state, the search process and smaller steps are done when most individuals are quite similar)

e. Mutation
Random mutation with a small independent probability. Generic algorithm combine an uphill tendency with random exploration and exchange of information among parallel search.

These algorithms are implemented for optimization and are commonly used for problems with a large search spaces [2] because this algorithms helps to find different solutions and mix them to generate new solutions with a bigger scope. GA are applied in a huge amount of problems, just to mention few of them [1], we have pattern recognition applications, robotics and artificial life applications, expert systems, electronic and electrical applications and different applications in Medicine and Biology.

3 Timetabling problem

This process is normally done manually and it can take even days to manage the assignment of resources. Due to the possible high data volume and the variables involved, this can be a very expensive task. The timetabling problem is defined as: ”The assignment, allowed by constrains, from a group of resources of time and space, to satisfy a set desired objectives”.[4] This is a classic problem used in the AI since it includes searching concepts with restrictions that could be implemented with several algorithms and it is considered as a NP-complete problem.

There is a need of organizing classes schedules from professors and students, all conditions are dynamically listed based on each school such as topics, availability of classrooms and professors, this cause a big issue in the successful planning and this particular problem is faced at the beginning of each period.

3.1 Problem statement

The proposed solution for the timetabling problem has restrictions and minimum requirements set. The restrictions are with the amount of classrooms with an specific capacity and specific classroom type, amount of professors with specific working hours and an specific number of classes. Below the numbers for each restriction that are used in the solution.

a. 10 classrooms, each classroom has a capacity and the equipment needed for the class (Chemistry lab, Computer lab, Biology lab, Normal classroom)
b. 10 available class hours, starting at 7:00 hrs and ending at 22:00 hrs.
c. 10 teachers with different shifts, where 3 of them have only morning availability (7:00-13:00), 3 of them have only night availability (17:30-22:00) and 4 of them have all day availability.
d. 100 different classes, assigned to the teachers, with the lab requirement (Chemistry lab, Computer lab, Biology lab, Normal classroom) and with a minimum classroom capacity requirement.
e. The classes must be from Monday to Friday
f. The 100 classes must be in the schedule without having 2 times the same class.
g. Professors cannot be assigned in the same day at the same time in different classrooms

3.2 Data representation

There are two key elements in the genetic algorithms and those are probably the most difficult part to define when we are implementing the algorithm. One is the representation of the problem which it is easier if it is a linear representation and the second one is the fitness function that will evaluate how good is the solution.

The restrictions state that we have 10 classrooms with 10 available hours and 5 days a week, which mean that we have 10 * 10 * 5 = 500 combinations of the 3 concepts. We are going to call as SLOT the combination of classroom, hour and Day (i.e. classroom a, hour 1, Monday). The representation of these 500 slots will be added to a Vector. This mean that slot 1 is on Monday, the first classroom at the first hour and the slot 100 is on Monday, the 10th classroom at the last hour. Even our solution is a cube where one side is the day, the second one is the classroom and the third one is the hour, we are representing it as a linear form, this will make the crossover easier.

The representation of the fitness function will be given with evaluation of each class assignment to a specific slot. We have 4 hard restrictions to evaluate:

- Validate if the classroom type is the needed for the class
- Validate if the number of students requested per class fits in the classroom
- Validate if the professor is assigned to the slot within the working hours specified for each one
- Validate if the professor is not assigned the same day at the same time to two different classrooms

Meanwhile we are evaluating each class assignment, we are going to sum one point per each one of the four restrictions listed if the class doesn't break the rule. So the fitness value will be calculated as:

$$\frac{Solution\_Score}{Maximum\_Score}$$

Solution_Score is sum of points earned

Maximum_Score is the number of classes * 4.

This means that if we have 100 classes, then we will need to get 400 points to have the right assignment, then a solution to the problem. Therefore the fitness value will be given with numbers between 0 and 1.

The restriction *f.* will be accomplished by a function in the algorithm that removes all the duplicated assignments to reassign the classes to a new slot.

For the classroom details, the representation of each classroom is given with the table 1

Table 1: Classroom details

Room	isChemLab	isCompLab	isBioLab	isNormLab	size
1	false	false	false	true	30
2	false	true	false	false	30
3	false	false	true	false	30
4	true	false	false	false	30
5	false	false	false	true	25
6	false	true	false	true	25
7	true	false	false	false	30
8	false	true	false	false	30
9	false	false	true	false	30
10	false	false	false	true	25

The 10 possible times for each class each day will be given in table 2.

Table 2: Hours details

Hour	Start_hour	End_hour
1	07:00	08:30
2	08:30	10:00
3	10:00	11:30
4	11:30	13:00
5	13:00	14:30
6	14:30	16:00
7	16:00	17:30
8	17:30	19:00
9	19:00	20:30
10	20:30	22:00

The professors restrictions, where the start hour is given by the hour time that the professor can start working and the end hour is when the professor must end the working hours. This information is given in table 3 using table 2.

The table 4 has an extract of all the different classes that we have and the structure that we used in the project. The slots were assigned to the different classes and the algorithm runs for the different slots.

3.3 Implementation

The implementation of the Algorithm was done using Java. As we said before, the key parts of a Genetic Algorithm consist in an initial population, a evaluation or fitness function, a crossover and a mutation. For this problem we added a specific

Table 3: Professors restrictions

Professor	start_hour	end_hour
1	1	4
2	1	4
3	1	4
4	8	10
5	8	10
6	8	10
7	1	10
8	1	10
9	1	10
10	1	10

Table 4: Classes definition extract

Class	Title	Professor	isChemLab	isCompLab	isBioLab	isNormLab	size
1	Literature 1	1	false	false	false	true	24
2	Literature 2	2	false	false	false	true	24
3	Mathematics 1	3	false	false	false	true	26
4	Mathematics 2	4	false	false	false	true	26
5	Mathematics 3	5	false	false	false	true	26
6	Mathematics 4	6	false	false	false	true	24
7	Introduction to Programming	7	false	true	false	false	26
8	Informatics 1	8	false	true	false	false	26
9	Informatics 2	9	false	true	false	false	26
10	Informatics 3	10	false	true	false	false	26
11	Algorithms 1	1	false	true	false	false	26
12	Algorithms 2	2	false	true	false	true	26
13	Chemistry 1	3	true	false	false	false	26
14	Chemistry 2	4	true	false	false	false	26
15	Chemistry 3	5	true	false	false	false	25
16	Biology 1	6	false	false	true	false	25
17	Biology 2	7	false	false	true	false	25
18	Biology 3	8	false	false	true	false	25
19	Finance 1	9	false	false	false	true	25
20	Finance 2	10	false	false	false	true	25

function, as part of the algorithm, to remove duplicated assignations. This was added because of the amount of items that were duplicated after each crossover. If we want to get a random selection of classes where no duplicates exist and no classes are missing, then this function is now a key part of the process. The genetic algorithm used can be found below as pseudo code:

```
public GA()
{
    createClassrooms();
    createProfessors();
    generateInitialPopulation();
```

```
        do
        {
            for i : this.TOTAL_POPULATION
            {
                if (curentSolution.fitnessValue >
                 bestSolution.fitnessValue) {
                    bestSolution = curentSolution.fitnessValue;
                }
                if (bestSolution.fitness_value == 1
                     || GenerationNumber > maxGenerations) {
                    print(solutionFound, generationNumber)
                    exit;
                }
            }
            orderPopulation();
            for i : this.TOTAL_POPULATION
            {
                randomSolutionSelection a,b;

                Chromosome child = crossover(a,b);
                newPopulation.add(child);
            }
//current population is now the new population
            initialPopulation = newPopulation;
            newPopulation = newPopulation.clear();
            numberOfGenerations++;
        } while (!complete);
    }
```

The algorithm shows how the initial population is handled and how is replaced with a new population created, but this new population based in the initial one. We have some functions within the code shown:

- createClassrooms(): It is done to generate the different classrooms that will be used with the different characteristics as shown in table 1.

- createProfessors(): As we created the classrooms, we need to generate the professors, specifically the shifts for each one as shown in the data representation section, table 3.

- generateInitialPopulation(): This function generates different solutions to the problem, the assignation between slots and classes is done randomly and it will generate as much solutions as we define in the TOTAL_POPULATION variable.

- orderPopulation(): Darwin said that the best individuals, who adapt better to the environment will survive. This method simulates that idea. Once we have the fit-

ness value for each solution, we perform an ordering for this solutions based in the obtained fitness value in order to be able to select just the best solutions for the crossover.

There is another function that was not explained before: crossover(a,b). This method is where we get the new solutions based on the two parents a and b. We can see how this crossover is performed with the following pseudo code:

```
input: parent a, b
output: child
    crossover(parent1, parent2) {
        newClasses = new Classes set;
        newClasses2 = new Classes set;

        //random number to divide parents
        int cut = randomNumber();

        for i : parent1.classes.size()
        {
            if (i <= cut) {
                newClasses.add(parent1.classes.elementAt(i));
                newClasses2.add(parent2.classes.elementAt(i));
            } else {
                newClasses.add(parent2.classes.elementAt(i));
                newClasses2.add(parent1.classes.elementAt(i));
            }
        }
        //assign the classes new solutions to two new children
        child.classes = newClasses;
        child2.classes = newClasses2;

        //remove duplicates assignment after crossover
        child = removeDuplicates(child);
        child2 = removeDuplicates(child2);

        if(random() > mutationProbability)
             child = mutate(child);
        if(random() > mutationProbability)
             child2 = mutate(child2);

        child.fitnessValue = fitnessFunction(child);
        child2.fitnessValue = fitnessFunction(child2);

        if (child.fitnessValue > child2.fitnessValue)
            return child;
```

```
        return child2;
    }
```

The crossover function contains also the mutation part of the algorithm and the fitness function. The mutation consist in taking a random class and change the slot assigned with a new one. This mutation pretend to simulate the evolution process where the individuals change to adapt better to environment, although sometimes this evolution is not the best change. Once a solution is mutated, it could be a better solution, but sometimes it couldn't. You will notice that the code as a mutationProbability conditional before performing the mutation. This is an important part of the algorithm, the change is not performed for every children generated, it depends in a probability given as parameter for the algorithm.

The fitness function, as we said, is based in a sum of points for each assignment. Each class assignment is evaluated with our 4 restrictions, if the class fulfills the 4 restrictions, then the solution gain 4 points. At the end of the evaluation, the amount of gained points is divided by the total possible points and we will get a number between 0 and 1. The goal is to get a solution that have 1 in the fitness value.

3.4 Results

There are general values that were common for the different tests we ran. In the first place, we already mentioned that the amount of classrooms is 10 and the amount of professors is also 10. The total population value for almost all the test was 100. There is one specific test where we moved the amount of the population just to compare the behavior and evaluate if the population is a relevant factor of the algorithm. The maximum generation number allowed was 1500, this number was a big number to let the algorithm run with different test cases, different initial populations and then evaluate the different results. The last common value is the mutation probability, it was set to 0.6 and the mutation depends on this number.

The first result is to show the behavior of the algorithm while we were increasing the amount of classes to assign into the slots. The image 1 shows how many generations were needed to find the solution for each amount of classes. We started with 7 classes until finishing with 100. It is important to mention that the number for each amount of classes is not just 1 run for each one, each value is an average of the results of 100 runs of the algorithm for each amount of classes. The image 2 shows the time in seconds used for each of runs for the classes, also it is an average of time of 100 runs for each class number. The exact values are shown in the Reference Test Table section. This first set of results are in table 5.

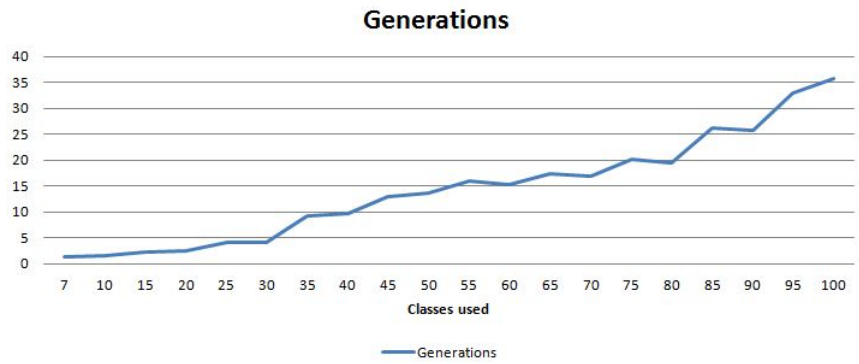


Fig. 1: Generations created per run

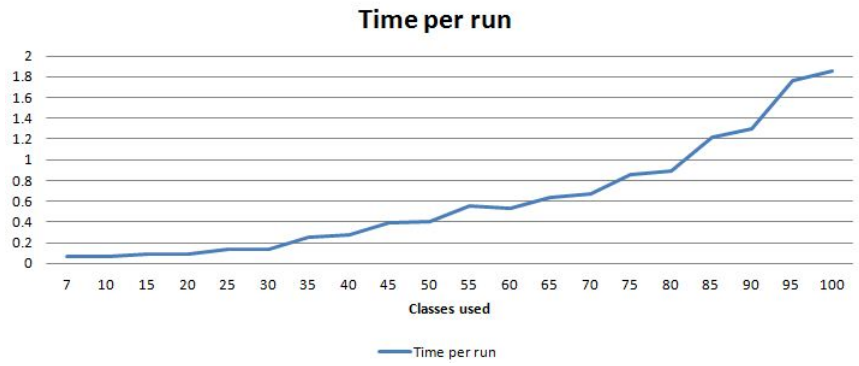


Fig. 2: Time needed per run

For both images, the results were the expected. It is normal to have more generations or more time used while we are adding classes to assign. It is important to mention that we had an issue when we started the tests. The issue was the mutation. At the beginning it was implemented as a random change of values, for the 7 class assignment, it worked, also for 10 even 20 classes. The problem started with 20 classes, because sometimes the algorithm didn't finish and the fitness value was around .85 of effectiveness. We faced the following challenge while we were analyzing the results for previous tests: The random re-assignment of values hardly took the wrong assignments, sometimes the selection was right but most of the time it was wrong. For a bigger number of classes the effectiveness was lower. This is not acceptable for this algorithm and this problem since the assignments must fulfill the restrictions, that is why they are called hard restrictions. The solution for this problem was changing the mutation function, with same characteristics but the random

assignment was just for those classes that were wrong. This made an execution time reduction in 90% and also the generations needed in 95%.

What about the population? Is it relevant to have a specific amount? The right answer is YES. It is very important to have the right number for the population. The only way to get the right number is with the experience, test with the problem and see where is the right value. In this case, with timetabling, it is important to see the behavior. If we see figure 3 we will note that we started with more than 500 generations needed to find the solution with 2 solutions in the population, but for 2 as population also we need to mention that we had 3 failed runs in the batch of 100, for the next tests we didn't had any failed result. As we are increasing the number allowed for the population, we reduce the number of generation needed, and you might say that if we increment the number of the population, then it will be faster, but take a look to the timing graph, figure 4, we will see a huge impact in the time needed for get the solution. That is not all, the figure 5 shows a zoom in in the first image just to see the data starting with the population number as 20. You will notice that it is a curve where the minimum value is around the population 100. This is the value that we decided to use for the different tests. The exact values used for the images are in the section Reference Test Tables table 6.

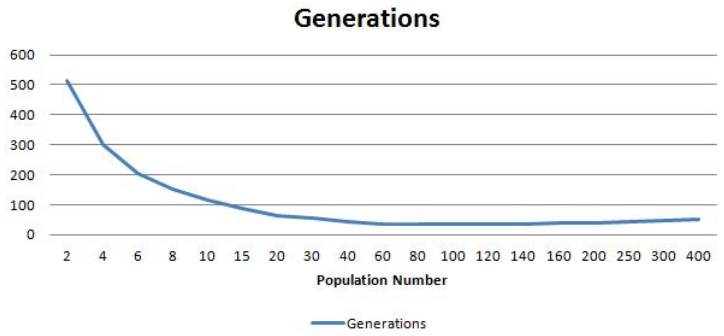


Fig. 3: Generations needed per run

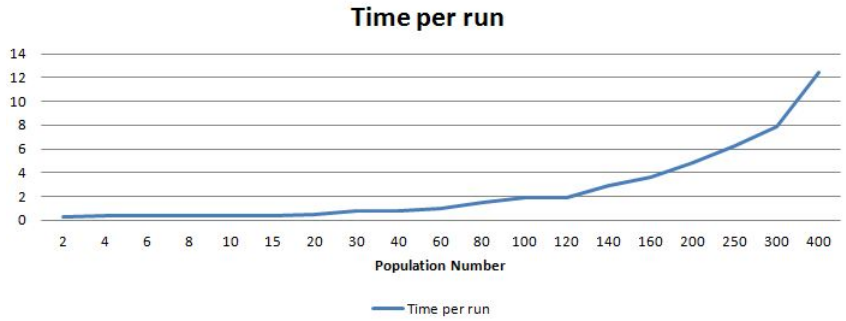


Fig. 4: Time needed per run

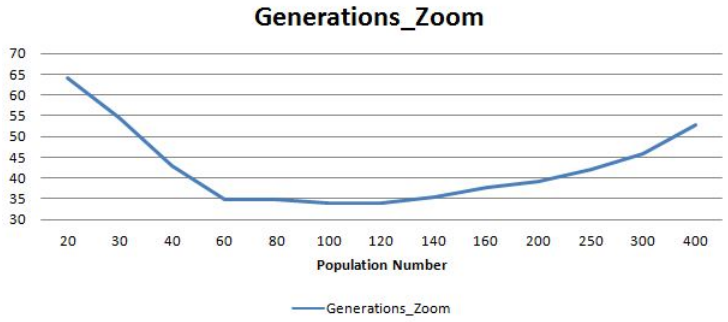


Fig. 5: Generations needed per run

Figures 6 and 7 show the progress of the algorithm for 50 classes and 100 classes. The bests solutions found are the ones shown in the image. Both get quickly to the 90% of right assignments and slowly go until it finds the goal.

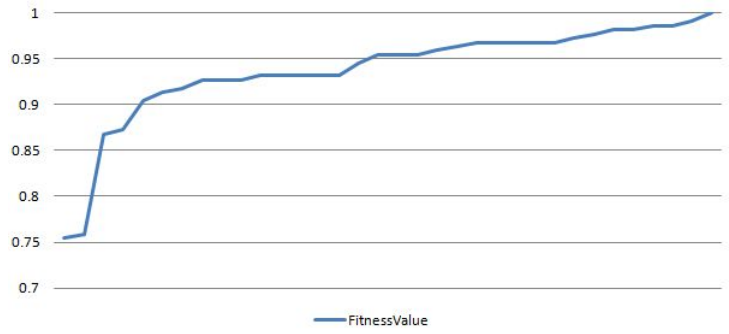


Fig. 6: 50 classes run progress

Fig. 7: 100 classes run progress

4 Conclusion

Genetic Algorithms are a great option for all the searching problems. You can be able to generate a big amount of different solutions in a very short time, review them and validate which are the ones that fulfill the restrictions. The complexity of GA is with the problem representation. If you don't have a good representation of the problem or if you miss any restriction you won't have the best solution or it could be wrong.

It is very important having the right way to evaluate a solution found. For any actual state of a problem, process, or result, if you don't have a way to measure it, then you won't have a way to improve it. With the Genetic Algorithms apply the same concept, if you don't have the right fitness function then it will be hard to find

the right solution. There are problems where it is hard to visualize a fitness function, but it must have the biggest effort while we are representing the problem.

Another quality we have with GA is the use of the probability. For example, when you work with the simulated annealing algorithm [3], you can get different solutions but you have the risk of stuck in a local maximum. If you add the probability, then you get better results, because you skip those local maximum and most of the time you get the results faster. It is the same case with GA, since you are using 2 different solutions, and both are in the group of the best solutions, then you use the probability for the mutation and you get the results in a good time, but this depend on the problem. We need to remember that this kind of problems are NP-Complete, the example we worked in this report is a small sub-conjunct of a big problem.

5 Reference Test Tables

Table 5: Test Results 1

Classes	Generations	Time/run
7	1.22	0.06431
10	1.45	0.0701
15	2.18	0.08675
20	2.41	0.09349
25	4.21	0.13323
30	4.15	0.13926
35	9.27	0.25726
40	9.63	0.27869
45	13.03	0.38757
50	13.56	0.40916
55	15.98	0.55404
60	15.26	0.53636
65	17.3	0.63967
70	16.79	0.67143
75	20.26	0.86235
80	19.36	0.88623
85	26.08	1.21973
90	25.82	1.30066
95	32.98	1.76559
100	35.61	1.85908

Table 6: Test Results 2

Population	generations	time/run
2	515.4123711	0.27528866
4	301.34	0.33909
6	204.99	0.35135
8	153.6	0.35486
10	115.34	0.34106
15	87.14	0.41051
20	64.05	0.43682
30	54.49	0.73711
40	42.94	0.79695
60	34.97	1.00219
80	34.71	1.45296
100	33.88	1.85956
120	33.88	1.85956
140	35.55	2.93996
160	37.73	3.6583
200	39.3	4.81199
250	41.94	6.2385
300	45.79	7.84993
400	52.91836735	12.46583673

References

[1] N. Chaiyaratana and A. M S Zalzala. "Recent developments in evolutionary and genetic algorithms: theory and applications". In: *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1997. GALESIA 97. Second International Conference On (Conf. Publ. No. 446)*. 1997, pp. 270–277. DOI: 10.1049/cp:19971192.

[2] David E. Goldberg. "Sizing Populations for Serial and Parallel Genetic Algorithms". In: *Proceedings of the Third International Conference on Genetic Algorithms*. George Mason University, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 70–79. ISBN: 1-55860-006-3. URL: http://dl.acm.org/citation.cfm?id=93126.93150.

[3] P Russell S.J.; Norvig. "Genetic Algorithms". In: *Artifitial Intelligence: A modern approach*. Madrid,2004: Pearson Education, 2004, pp. 131–135. ISBN: 978-84-205-4003-0.

[4] Anthony Wren. "Scheduling, timetabling and rostering - A special relationship?" In: *Practice and Theory of Automated Timetabling*. 1996, pp. 46–75. DOI: 10.1007/3-540-61794-9_51.

Analysis of Intrusion Detection System Using Multilayer Perceptron Neural Networks

Mariana Galván Elvira

Abstract Nowadays, network security is involved in organizations, enterprises, and other types of institutions and it is essential to protect business's reputation. Networks need to be protected against intruders; one way to monitor your network is the use of an Intrusion Detection System (IDS). Various techniques have been applied but those are not effective, this paper provides an analysis of different neural network techniques using DARPA dataset and Neuroph Java Neural Network Framework tool.

1 Introduction

1.1 Definition

This paper presents a neural network-based intrusion detection method for the internet-based attacks. Neural networks are used to identify and predict unusual activities in the system. According to the CERT (Computer Emergency Response Team) [1], the number of reported incident has increased. An Internet attack can be defined as the misuse of a system or systems without permission. The person performing the attack is called a hacker or cracker. These intruders can be classified into two categories, outsiders and intruders.

Attacks can be classified into three types[5]:

- Reconnaissance: involve the gathering of information about a system in order to find its weaknesses.
- Exploits: take advantage of a known bug or design flaw in the system.
- Denial-of-Service (DoS): disrupt or deny access to a service or resource.

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201.

1.2 Types of Attacks

DARPA [2] categorizes the attacks into five major types based son the goals and actions of the attacker:

- DoS attacks try to make services provided by user to be restricted or denied. Example: SYN-Flood attack.
- Probe attacks attempt to get information about an existing computer or network configuration.
- Remote-to-local (R2L) attacks are caused by an attacker who only has remote access rights.
- User-to-root (U2R) attacks are performed by an attacker who has right of user level access and tries to obtain super user access.

1.3 Actual Intrusion Detection Systems

Intrusion Detection Systems (IDS) monitor the network for any unusual activity. The primary aim of Intrusion Detection Systems is to protect the availability, confidentiality and integrity of critical networked systems.

There are two types of IDS:

- A host-based: it runs on a single machine and monitors its own traffic for attack.
- Network-based: it is located on an independent machine watching the activity of the entire network.

Intrusion can be detected using two different methods:

- Misuse intrusion detection: it compares the network traffic against a database of known attacks. An alarm is set off when an event matches the signature of an attack in the database.
- Anomaly intrusion detection: It analyses the network traffic for any deviation from the normal or expected behavior of the system. It then learns and adapts from that information.

The high return of false positive is the main disadvantage of the anomaly intrusion detection system. The disadvantage of misuse intrusion detection is keeping the database up-to-date.

Different techniques have been applied to the data that reaches the Intrusion Detection System. Most common approaches below[6]:

- Expert Systems
- Signature Analysis (This is one of the most common methods used in commercial systems such as Stalker, Real Security, and Cisco IDS)
- Colored Petri Nets
- Statistical Analysis
- Data Mining

- Neural Networks

Several neural networks-bases approaches have been employed for intrusion detections. The recent rapid development in data mining has made available a wide variety of algorithms[3]; the most popular algorithm for mining rules based on two-valued attributes is APRIORI.

Applying fuzzy methods for the development of IDS provide some flexibility to the uncertain problem of intrusion detection. Most of the fuzzy IDS require human experts to determine the fuzzy sets and set of fuzzy rules.

A particle swarm optimization (PSO) is a search algorithm based on birds group behavior and biological population model[7], it is robust and has been proven theoretically and empirically to be able to search the optimum solution or near-optimal solution to a complex problem. Particle swarm optimization algorithm is also a new optimum algorithm developed fast recently, it has some advantages such as the parallel search and the searching efficiency is higher.

In recent years, the research on intrusion detection is gradually inclined to artificial intelligence technology to improve the detection accuracy. Among the rest, the most representative intelligent method is neural network.

2 Neural Networks

2.1 Introduction

Neural newtorks are predictive models based on the action of biological neurons. This name was one of the great successes of the Twentieth Century. A typical neural network might have a hundred neurons.

The original “Perceptron” model was developed by Frank Rosenblatt in 1958[4]. This model consisted of three layers: inputs, association units and output layer. Unfortunately, the use of a step function in the neurons made the perceptions difficult or impossible to train. The critical analysis of perceptrons was published in 1969 by Marvin Minsky and Seymore Papert.

Interest in neural networks was revived in 1986 when David Rumelhart, Geoffrey Hinton and Ronald Williams proposed a multilayer neural network with nonlinear but differentiable transfer functions.

2.2 The Multilayer Perceptron Neural Network Model

The following diagram ilustrates a full-connected, three layer, feed-forward, perceptron neural network:

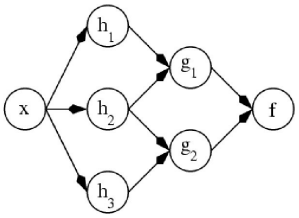


Figure 1. A Multilayer Perceptron Neural Network Model

All neural networks have an input layer and an output layer, but the number of hidden layers may vary. The training process of a multilayer perceptron network is to find the set of weights values that will cause the output from the neural network to match the actual target values as closely as possible.

One of the most important characteristics of a perceptron network is the number of neurons in the hidden layer(s), if an inadequate number of neurons are used, the newtork will be unable to model complex data. If too many neurons are used, the network may over fit the data.

2.3 Learning through backpropagation

As mentioned, learning occurs by changing connection weights after changing connections weights after each pice of data is processed. Backpropagation is a supervised learning method that requires a dataset of desired output for many inputs, making up the training set.

The backpropagation learning algorithm can be divided into two phases:

- Propagation: forward and backward propagation through the neural network in order to generate the propagation's output activations and generate the deltas of all output and hidden neurons.
- Weight Update: multiply its output delta and input activation to get the gradient of the weight, subtract a ratio of the gradient from the weight.

These phases are repeated until the performance of the network is satisfactory, the algorithm for a 3-layer network is shown below:

```
1  initialize network weights (small random values)
2        do
3                forEach train example exVpredict = neural-net-output(network,ex)
4        actual = teacher-output(ex)
5                compute error ( prediction - actual ) at the output units
6                compute D_w_h for all weights from hidden layer to output layer
7                compute D_w_i for all weights from input layer to hidden layer
8                update network weights
9        until all examples classified correctly or another stopping criterion satisfied
10 return the network
```

3 DARPA Dataset

3.1 Introduction

The MIT Lincoln Laboratory has collected the first evaluation dataset for evaluation of computer network Intrusion Detection Systems (IDS)[2]. This dataset is used for the purpose of training and testing. All the network traffic included was recorded in tcpdump format and provided for evaluation. The dataset consist of weeks one, two and three of training data and weeks four and five of test data.

3.2 Data Description

Data used on this paper was made available in March 1998. A set of attacks and list of anomalies were selected to train and test the neural network. The following table describes all attacks included in training data that has been posted to the Lincoln Laboratory web site[2]:

Attack	Description
back	Denial of service attack against apache webserver
eject	Buffer overflow for a valid user using simple variants of the account name
format	Buffer overflow using the fdformat UNIX system
ftp-write	Remote FTP user creates .rhost file in world writable anonymous FTP
guest	Try to guess password via telnet for guest account
ipsweep	Surveillance sweep performing either a port sweep or ping on multiple addresses
land	Denial of service where a remote host is sent a udp packet.
neptune	Syn flood denial of servie on one or more ports
warez	User logs into anonymous FTP site and creates a hidden directory
smurf	Denial of service icmp echo reply flood

Table 1. Attack list during training and testing processes

3.3 Dataset Format

Data was provided for audit data collected using Sun's Basic Security Monitoring (BSM) software[2]. Each line in a list corresponds to a separate sessions. Each session corresponds to an individual TCP/IP connection between the beginning of the initial three-message hanshake to the final FIN and ACK sequence.

A session is uniquely specified by the start time, by the source and destination IP addresses, and by the source and destination ports. The eight columns in list files provide information which identifies the TCP/IP conection. The performance

of intrusion detection systems will be evaluated using scores assigned to the ninth column, the last column is optional and can be filled with an attack name.

The columns in the list file contain the following:

Column	Description	Example
1	Date	01/27/1998
2	Starting time	08:45:01
3	Duration	00:00:01
4	Service	finger
5	Source port	1050
6	Destination port	79
7	Source IP address	192.168.0.40
8	Destination IP address	192.168.1.30
9	Score	1
10	Attack	guess

Table 2. Data session connection example

4 Neuroph Java Neural Network Framework

Neuroph is a Java neural network framework to develop common neural network architectures [**Neu00a**]. Also includes a GUI neural network editor to create Java neural networks components, some of the important features of the Neuroph framework are:

- Adaline
- Perceptron
- Multi Layer Perceptron with Backpropagration
- Hopfield Network
- Bidirectional Associative Memory
- Neuro Fuzzy Reasoner

Neuroph Java Neural Network Framework also support supervised and unsupervised learning rules, data normalization and simple microbenchmarking framework. For our testing purposes, we will do some tests using the Neuroph Java Neural Network Framework tool.

5 Methodology

In this work, a multilayer perceptron neural network based on the backpropagation training algorithm is used. The architecture proposed is as follows:

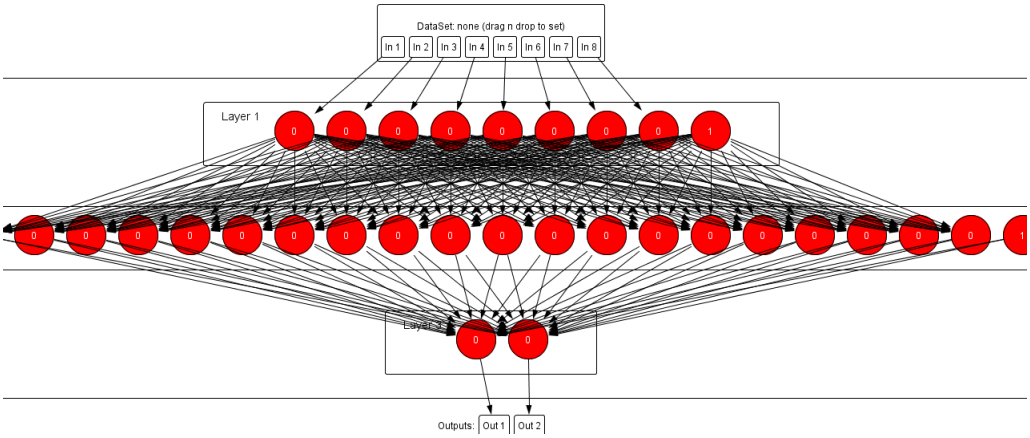


Figure 2.- Architecture proposed

Described below the steps and configuration implemented:

- Randomly select proper data and anomaly data using DARPA dataset.
- Create a training and analysis list, obtaining main cases and then normalize the data.
- Use the proposed intrusion detection model using Neuroph Java Neural Network Framework.
- Train the neural network.
- Execute anomaly detection.

	NN1	NN2	NN3
Input Neurons	8	8	8
Output Neurons	2	2	2
Hidden Neurons	20	20	20
Transfer Function	Sigmoid	Tanh	Sigmoid
Learning Language	Backpropagation	Backpropagation	Dynamic Propagation
Max Error	0.001	0.001	0.001
Learning Rate	0.02	0.02	0.02

Table 3. Neuroph Java Neural Network Configuration

6 Experiments

Three experiments have been conducted, described below these results:

Input	Total Main Square Error	Desired Output
1	0; 0.009	0; 0
2	1; 0.9718	1; 0.0588
3	0; 0.012	0; 0
4	1; 0.7528	1; 0.7059
5	0; 0.0934	0; 0
6	1; 0.9938	1; 0.8235
7	0; 0.9983	0; 0
8	0.9995; 0.6578	1; 1
9	0.2167; 0.0041	1; 1
10	1; 0.9969	1; 1
11	1; 0.9992	1; 1
12	1; 0.0018	1; 0.3824

Table 4. Neural Network Test Results

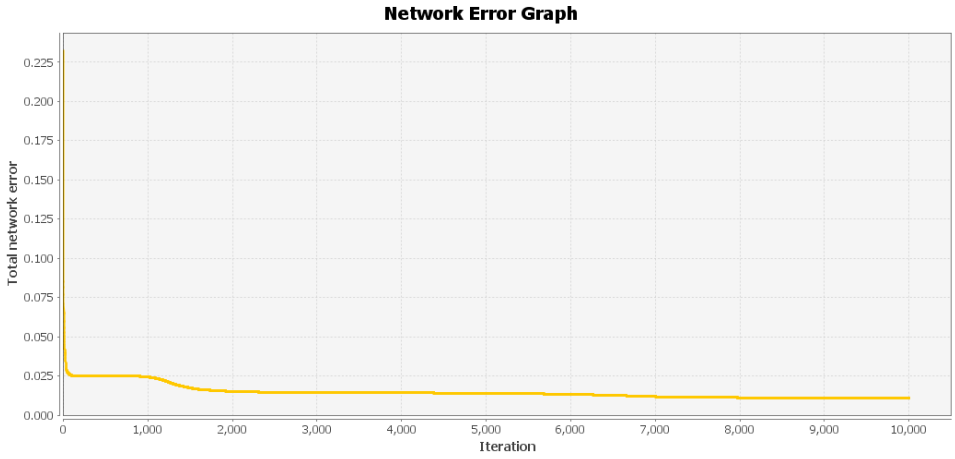


Figure 3. Network Error Graph of Neural Network

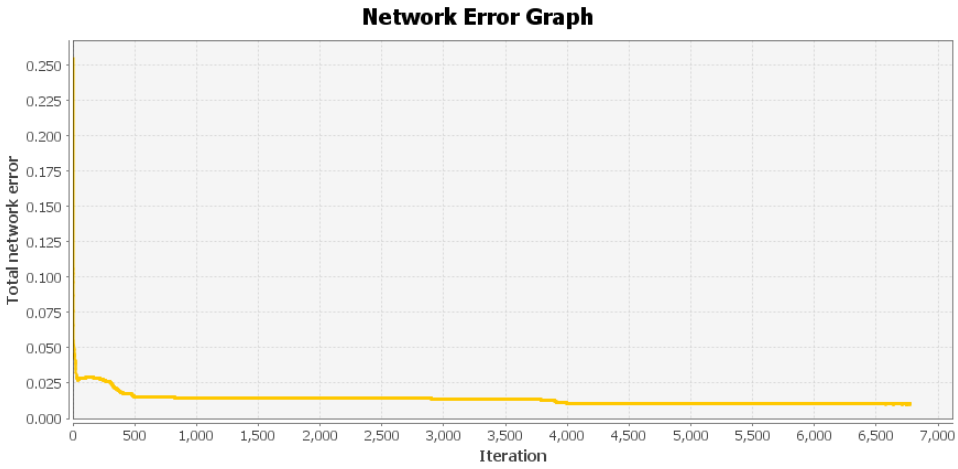


Figure 3. Network Error Graph of Neural Network with Momentum

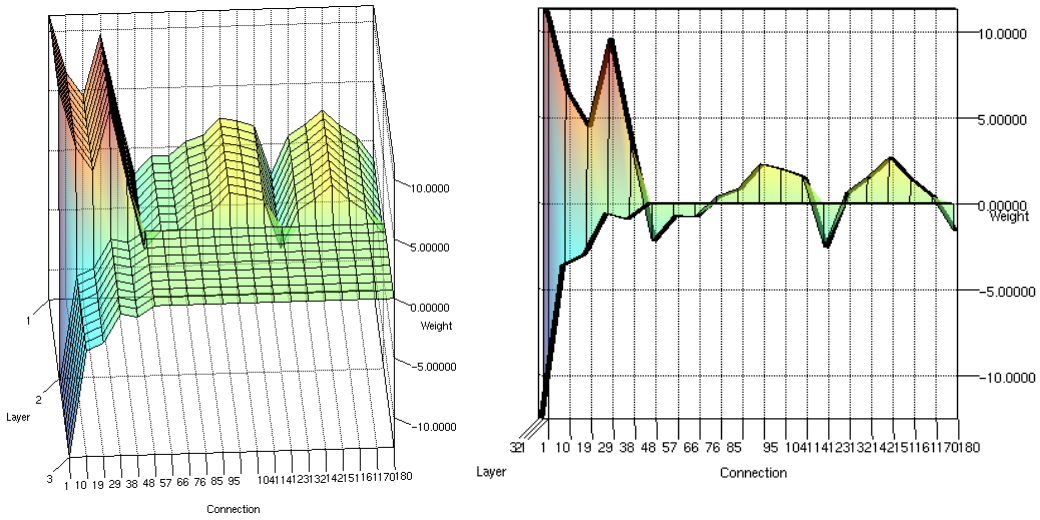


Figure 4 & 5. Weights surface 3D, Relation between connections and weights

7 Conclusions

Intrusion Detection is one of the main topics discussed of the network security. Artificial Neural Network suffers from numerous drawbacks, such as, difficult to determine architectural parameters, local minima, over fitting. Both simulation and experiment indicate that the Multilayer Perceptron Neural Network was able to identify attacks but not to classify them, the results also indicate that the neural network implemented was able to classify 67% of unknown attacks. Including a hybrid system could be an option for aiding network systems in the task of computer intrusion detection.

References

[1] "CERT Computer Emergency Response Team". In: (www.cert.org). URL: www.cert.org.

[2] "DARPA intrusion detection evaluation". In: (http://www.ll.mit.edu/IST/ideval/data/data index.htm). URL: http://www.ll.mit.edu/IST/ideval/data/dataindex.htm.

[3] Ertoz L Dokas .P. "Data Mining for Network Intrusion Detection". In: (2002).

[4] "Multilayer Perceptron Neural Networks". In: (http://www.dtreg.com/mlfn.htm).

[5] "Neuroph Java Neural Network Framework". In: (http://www.infosyssec.net/infosyssec/netintrufaq.htm). URL: http://www.infosyssec.net/infosyssec/netintrufaq.htm.

[6] Jimmy Shum and Heidar A. Malki. "Network Intrusion Detection System Using Neural Networks". In: ().

[7] JiChen Liu WenJie Tian. "Network Intrusion Detection Analysis with Neural Network and Particle Swarm Optimization Algorithm". In: ().

Implementación de un sistema experto basado en sistemas de inferencia difusa para el cálculo de consumo calórico

Ricardo Díaz

Abstract El presente trabajo muestra el desarrollo e implementación de un sistema experto basado en sistemas de inferencia difusa (FIS) para la determinación del consumo calórico diario de una persona a partir de los parámetros biómetricos de: índice de complexión (ICC) corporal, porcentaje de actividad física e índice de masa corporal (IMC). El sistema se implementa en Matlab con la herramienta de lógica difusa. Se configuró el FIS con funciones de pertenencia trapezoidales para cada parámetro a considerar; su sistema de reglas de decisión es establecido con base en el conocimiento y experiencia de un profesional del área de estudio.

1 Introducción

Los sistemas expertos basados en sistemas de inferencia difusa han sido implementados en el área de la salud principalmente para la simulación del conocimiento y experiencia en la toma de decisiones médico-clínicas en pacientes bajo condiciones fisiológicas adversas.

Definiciones

- ***Sistema experto:*** *programa computacional que permite simular la forma en que un especialista toma una decisión contemplando determinada información [***tesisgisela***]. El nivel de precisión con el cual cuentan los sistemas expertos depende primordialmente de la experiencia del especialista, el conocimiento enfocado al área de interés y la calidad de la información.*

- ***Lógica difusa:*** *técnica de inteligencia artificial que brinda la posibilidad de manejar información con alto grado de imprecisión. Es considerada una lógica*

Tecnologico de Monterrey, Campus Guadalajara. Av. General Ramon Corona 2514, Zapopan, Jalisco. C.P. 45201. , e-mail: r.diaz@itesm.mx

*multívaluda que permite la existencia de valores intermedios para definir conceptos que no son complementen falsos ni completamente verdaderos. En la teoría de conjuntos un elemento puede pertenecer o no a un conjunto, sin embargo en los conjuntos difusos estos niveles de pertenencia se establecen a través de un valor numérico entre 0 y 1, donde 1 indica total pertenencia y 0 total no pertenencia. [***tesisgonzalez***]*

Un sistema experto basado en sistemas de inferencia difusa tiene una estructura básica de tres elementos [**tesisgisela**]

- **La base del conocimiento:** donde se incluye el conocimiento específico. Para el correcto funcionamiento el sistema experto debe contar con el mismo conjunto de variables con las cuales cuenta (o contaría) un especialista para tomar una decisión. La forma más común establece una relación, reglas, de tipo IF-THEN
- **Sistema de inferencia:** es la forma en que se relacionan cada una de las variables de decisión a través de un sistema de reglas establecidas por el especialista.
- **Interfaz de usuario:** elemento que permite la interacción entre el usuario y el sistema.

La generación de un sistema experto para la determinación del consumo calórico es importante ya que la sana alimentación, basada en un consumo calórico adecuado, es uno de los factores principales que contribuyen en la salud de las personas. Cada uno de los rangos de los parámetros a considerar para determinar el consumo calórico diario han sido establecidos por organizaciones de corte internacional para su estandarización, sin embargo en este tipo de clasificaciones se considera un somatotipo distinto al que predomina en México

Cuando se pretende establecer el consumo calórico de una persona es difícil considerar el uso de fórmulas generales o consumos preestablecidos puesto que, éstos, sólo consideran parámetros generales y no permiten tomar en cuenta la percepción de un especialista, quien toma una decisión con base en la práctica, la observación y la combinación de la diferentes variables biométricas que presenta un paciente.

2 Metodología

Con base en la experiencia de un especialista se establecen los parámetros de mayor relevancia para el cálculo de consumo calórico diario:

- Índice de masa corporal (IMC)
- Índice de complexión corporal (ICC)
- Porcentaje de actividad física)

De acuerdo a la Organización Mundial de la Salud (OMS)[**oms**] los rangos estándar para cada uno de los parámetros que se deben considerar para determinar

el consumo calórico pueden/deben ser modificables en función de la particularidad de cada persona [**ens**]. Por lo anterior se establecen rangos con algunas modificaciones, basadas en la experiencia, que permitan simular en el sistema experto el conocimiento del profesionista. Los parámetros biométricos a considerar se calculan de la siguiente manera.

2.1 IMC

El índice de masa corporal se obtiene de la siguiente formula[**ens**]:

$$IMC = \frac{m}{h^2} \tag{1}$$

Donde:
m= masa de la persona en kilogramos (kg)
h=altura de la persona en metros(m)

2.2 ICC

El índice de complexión corporal de una persona se determina a través de la siguiente formula[**ens**]:

$$ICC = \frac{h}{CM} \tag{2}$$

Donde:
m= masa de la persona en kilogramos (kg)
CM=circunferencia de muñeca (m)

2.3 Actividad Física

Para determina el cálculo de la actividad física de una persona es necesario clasificarlos con la tabla 1[**ens**]:

Tabla 1, Porcentaje de actividad física

Porcentaje Act. Física.	Clase
10 al 20	**Sedentaria**
30 al 40	**Normal**
40 al 50	**Moderado**
50 o más	**Alto**

Para determinar la clasificación de la actividad física se deben considerar los siguientes rangos basado en consumo calórico semanal:

- **Sedentario:** consumo calórico semanal menor a 1000 calórias
- **Normal:** consumo calórico semanal mayor a 1000 calórias y menor a 2000 calórias
- **Moderado:** consumo calórico semanal mayor a 2000 calórias y menor a 3000 calórias
- **Alto:** consumo calórico semanal mayor a 3000 calórias

Para la implementación del FIS se hace uso de funciones de pertenencia (de tipo trapezoidal); mismas que están definidas por los limites inferior a, superior d y los límites de soporte inferior b y superior c tal que $a < b < c < d$ [**tesisgonzalez**] . Esto se puntos se representa en la Figura 1

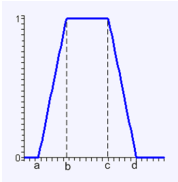


Fig. 1: Función de pertenencia Trapezoidal

Para definir los limites y soportes de la funciones de pertenencia de cada parámetro, se presentan los valores definidos por el experto . La clasificación se muestra en las tablas 2-4.

Tabla 2, Índice de masa corporal ajustado para los límites de la función trapezoidal

IMC	Inferior	Intermedio	Superior
Bajo	18 o menos	18 o menos	19
Normal	18	19 -24	25
Sobrepeso	24	25 - 29	30
Obesidad	29	30 o más	30 o más

Tabla 3, Índice de complexión corporal ajustado para los límites de la función trapezoidal

ICC	Inferior	Intermedio	Superior
Alta	9.6 o menos	9.6 o menos	10
Mediano	9.6	10.1 -10.4	11
Bajo	10.4	11 o más	11 0 más

Tabla 4, Actividad Física ajustado para los límites de la función trapezoidal

Act. Física	Inferior	Intermedio	Superior
Sedentario	10 o menos	10 o menos	15
Normal	10	15 - 20	25
Moderado	20	25 - 30	35
Alto	30	35 o más	35 o más

Con la caracterización de los parámetros se establece una matriz de decisión para determinar el consumo calórico en términos linguísticos (igualmente definidos por el experto); mismos que permiten determinar la cantidad de calorías recomendadas para una persona. La tabla de decisión se muestra en la Figura 2

Ejercicio	Sedentario			Normal			Moderado			Alto		
Complexión	Baja	Mediana	Alta	Baja	Mediana	Alta	Baja	Mediana	Alta	Baja	Mediana	Alta
Bajo	Bajo1	Bajo3	Bajo2	Bajo3	Bajo4	Bajo4	Medio1	Medio2	Medio2	Medio3	Medio4	Alto1
Normal	Bajo2	Bajo3	Bajo4	Bajo4	Medio1	Medio2	Medio2	Medio3	Medio4	Alto1	Alto2	Alto3
Sobrepeso	Bajo1	Bajo2	Bajo3	Bajo4	Bajo4	Medio1	Medio2	Medio2	Medio3	Medio4	Alto1	Alto2
Obesidad	Bajo3	Bajo3	Bajo4	Bajo4	Medio1	Medio2	Medio2	Medio3	Medio3	Alto4	Alto4	Alto4
IMC												

Fig. 2: Tabla de decisión considerando los parámetros biométricos

3 Implementación

Para la implementación del sistema experto es necesario definir las características del mismo. Este es un sistema de tipo Mandani, el cual se caracteriza por tener entradas y salidas no difusas, por lo que es necesario que se coloque un fuzificador y un defuzificador respectivamente [**tesisgisela**] (el sistema de defuzificacón es de tipo centroide). Su base de reglas está definida por expresiones lingísticas que tienen un antecedente para una consecuencia determinada (regla IF-THEN). Para generar el sistema experto se utiliza el toolboox Fuzzy de Matlab en el cual es posible desarrollar de manera gráfica el sistema experto.

Se implementa en el toolbox de Matlab un sistema con las tres variables de entrada correspondientes al: IMC, ICC y porcentaje de actividad física, Figura 3

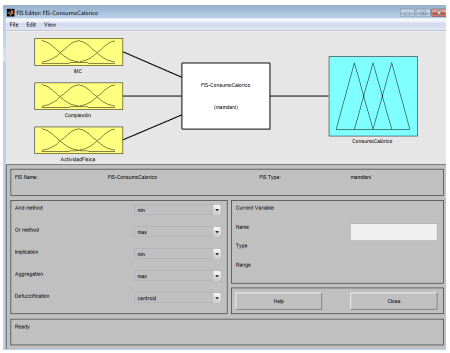


Fig. 3: FIS con las tres variables del sistema

Cada una de las variables debe ser implementada en las funciones de pertenencia que los clasifica. Con base a los estándares (rangos) establecidos en las tablas dos a cuatro. La implementación en Matlab se presenta en la Figura 4

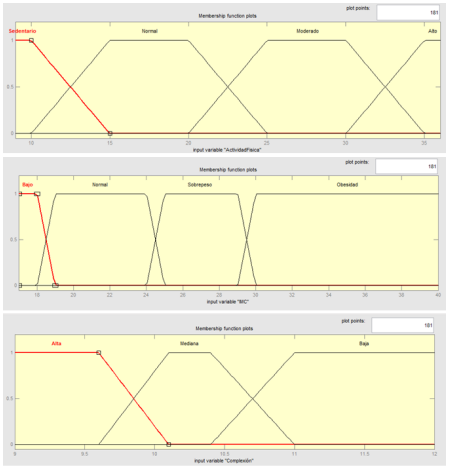


Fig. 4: Funciones de pertenencia de los parámetros, de arriba a abajo, porcentaje de actividad física, IMC y ICC

Debe mencionarse que el cambio en los valores del consumo calórico, a términos linguísticos, presentes en la tabla 2 basa el valor cuantitativo de las calórias en la Tabla 5

Tabla 5, Consumo calórico en términos linguísticos

Clase	Calorías
Bajo1	1400
Bajo2	1500
Bajo3	1600
Bajo4	1700
Medio1	1800
Medio2	1900
Medio3	2000
Medio4	2100
Alto1	2200
Alto2	2300
Alto3	2400
Alto4	2500

Considerando lo establecido en la tabla 2, en la herramienta de lógica difusa se puede crear el conjunto de reglas (conocimiento) de manera gráfica; basta seleccionar una clasificación de cada parámetro y selecciona el consumo calórico linguístico asignado. Se utilizan reglas de tipo IF-THEN

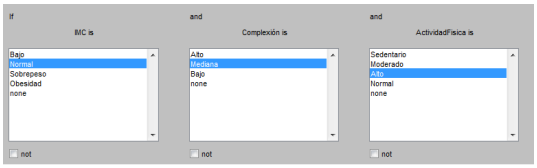


Fig. 5: En la interfaz gráfica se seleccionan los parámetros de acuerdo a lo establecido en la figura 2 a ellos se le asigna un valor de la tabla 5

Una vez cargado el sistema de reglas es posible generar las superficies de desición. Por ser un sistema experto basado en lógica difusa de tres variables, el programa permite graficar el comparativo entre dos de ellas. Las superficies de IMC contra ICC, figura 6, y la superficie IMC vs porcentaje de actividad física, figura 7, permiten visualizar la manera en que el sistema recrea el conocimiento del experto

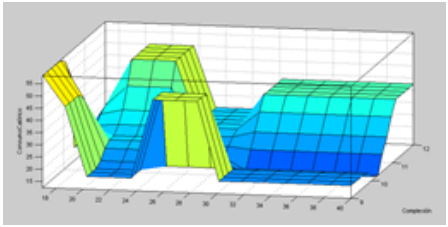


Fig. 6: Superficie IMC vs ICC

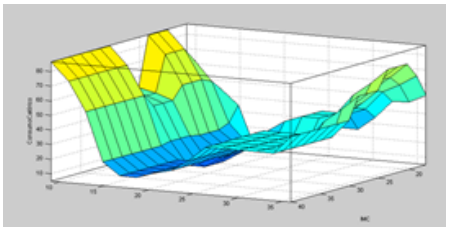


Fig. 7: Superficie IMC vs Porcentaje de Actividad Física

4 Resultados

Para la evaluación del sistema experto se realizó una prueba con 8 sujetos. De cada uno de ellos se obtuvó el IMC, ICC y la actividad física. Para tener un comparativo que defina el consumo calórico se realizó el calculo con un estándar establecido. Ademas del valor estimado por el experto y el valor aproximado del FIS se obtuvieron los valores aproximados de consumo calórico utilizando la ecuación de Harris-Benedict [**harris**]. La Tabla 6 presenta el comparativo de los parámetros mencionados.

Tabla 6, Comparativa entre el consumo calórico calculado por el índice Harris-Benedict, el experto y el FIS

#	IMC	% Act. Física	ICC	Harris-Benedict	Experto	FIS	%Error
1	31,1	10	15	1646.6	1600	1530	4.3
2	19.5	10	15	1603.2	1700	1510	11.1
3	27.3	9.6	20	1983.8	1900	1950	12.6
4	20.7	11	30	1999.6	2000	1950	2.5
5	27.6	11	15	1762.2	1700	1540	9.4
6	24.6	11	25	2103.7	2000	1860	7.0
7	21.6	10.4	25	1763.3	1750	1720	1.7
8	33.2	10.75	14	2268.5	2200	1910	13.1
						Promedio	**6.49**

5 Análisis y discusión

En la tabla 6 se calcula el error entre el valor estimado por el experto y el valor aproximado por el FIS. En ella se puede corroborar que el 75% de los datos tiene un error por debajo del 10%, límite considerado aceptable para la deteminación de consumo calórico diario. El 25% de los datos sobrepasa en 2% (+- 1%) el 10%, rango

de tolerancia establecido. De la comparación entre el valor estimado por el experto y la formula de Harris-Benedict se puede observar que la mayor parte de los datos presentan variaciones de +- 60 calórias en promedio; de ello se debe mencionar que el calculo del experto fue considerando los mismos parḿetros FIS y sin necesidad de obtener los cálculos que se deben realizar en la ecuación de Harris-Benedict.

6 Conclusión

El sistema experto basado en sistemas de inferencia difusa cuenta con un error promedio del 6.5%, margen de error que se encuentra por debajo de lo establecido en los estándares nacionales [**ens**] para una correcta determinación de la cantidad de calorías de consumo diario. Finalmente es importante resaltar que el FIS brinda una aproximación bastante confiable en relación con los valores estimados por la experta, misma que aproxima el valor del consumo con error menor al 10% comparado con la ecuación de Harris-Benedic.

Agradecimientos

Se reconoce y agradece a la Licenciada en Nutrición. Crystal Camacho Vélez por brindar su conocimiento y experiencia para el desarrollo de este trabajo.

References

1. S. Gisela, *Sistemas Neurodifusos adaptables y su implementación en el área biomédica*. México: Tec de Monterrey, campus Guadalajara, 2012.
2. G. Jhonny, *Sistema experto basado en sistemas de inferencia difusa para la evaluar la condición operativa de celdas de reducción de aluminio*. Colombia: Escuela de Ingeniería de operaciones,2008.
3. OMS, *Una guía de enfoques basados en población para incrementar los niveles de actividad física*. Ginebra, suiza: Organización Mundial de la Salud, 2008.
4. S. Claudia,V.Oscar, B. Arturo *200 Working group anthropometric cutoff points for predicting chronic diseases in the mexican national health survey 2000*. México: Encuesta Nacional de Salud, 2003.
5. M. Allan *The Harris Benedict equation reevaluated: resting energy requirements and the body cell mass*. Méxicoa: American Journal Of Clinical Nutrition, 1984