Tec de Monterrey Campus Querétaro

Arturo Rivera Flores - A01202457

Cuauhtemoc Suarez - A01206503

Francisco Villanueva - A01202413

Artificial Intelligence

15/11/16

**Lab 7: Implementing Neural Networks**

The WEKA software was also used for the previous Laboratory.

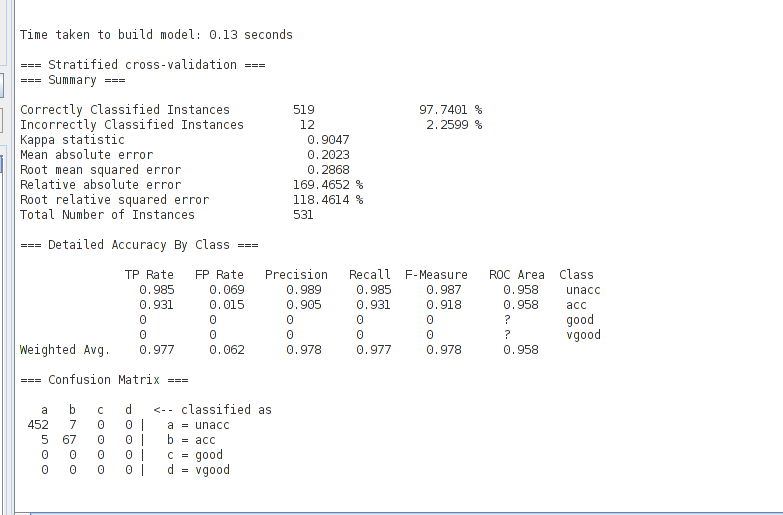
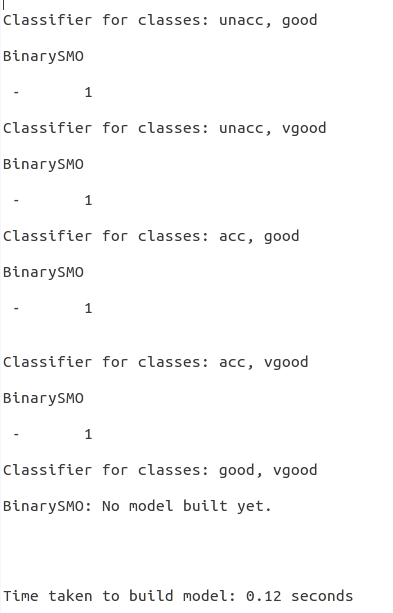


Figure 1.1 Results of cars.arf data set

For testing the capabilities of the multi-layered ANN we used a data set that was previously used for the decision trees, it is the cars.arf data set. It consists in 531 rows of data with 4 fields for determining if a Car is classified as unacc, acc, good, vgood.

We used a simod algorithm for this case, we can't use the voted Perceptron because of all the variables that the arff has. This created 5 neurons.



Weka did the analysis of the data in less than 0.12 seconds and the percentage of correct classifications were of more than 97% being pretty accurate.

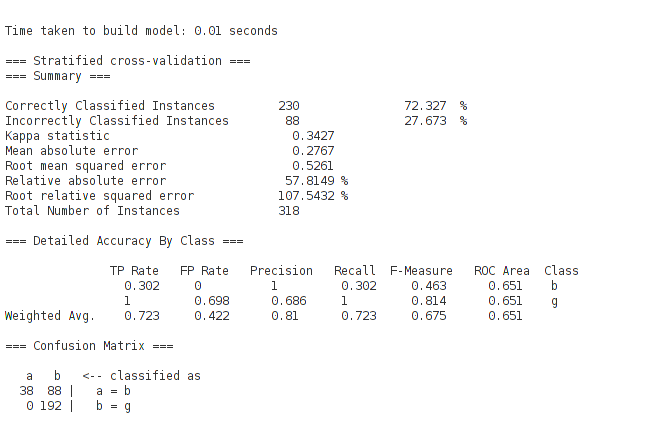


Figure 1.2 Results with Boolean data set

Then, the data set was changed with one that uses only Boolean values. The data set has 318 rows and only two fields. Weka finished the analysis in less than 0.01 seconds but the accuracy was of 72.32%

The used problems are categorized as classification problems so one could think that WEKA is using a step function as its activation function. The learning is most likely to be by back propagation as we are giving the inputs and the expected outputs at the beginning. Moreover, the first data set was previously tested and with the given information the classifications could be done without confusion (no two entries with the same fields gave different outputs), for this data set there could be entries with same fields but different outputs introducing the noise for the ANN not reaching the desired answer.

In general, ANN provide various advantages over other methods previously discussed in class as it need not to know the attached probabilities or statistics to solve the problem, it starts learning on its own by back propagation. As a matter of fact, the problem is that the process of learning is too slow and the number of variable parameters (learning rate, threshold, epoch, number of layers, number of neurons, etc.) is high so tuning correctly an ANN is difficult for reaching the desired result and avoiding overfitting.

The applications that an ANN can have depends on the structure of the network. For a multi layered simple structure (without being a closed loop), the problems that it can tackle are those about classification or linear regression, this could be use for obtaining the right parameters of a polynomial equation that maximizes the input (or minimizes), other uses are image recognition, pattern recognition, etc. Furthermore, while investigating closed loop ANNs we found that speech recognition can be done with this kind of networks; other uses are for diagnosing diseases.

Lastly, the problems of time complexity and the overall complexity (implementing the correct network) need to be considered as there are applications where other methods could outperform an ANN easily. Moreover, a lot of data and computational power is used for an ANN so the benefits need to overcome the difficulties for the network to be useful, in this cases it is worth implementing and training the ANN.

* Code Appendix

A part of the lab was implementing a perceptron and training it for a classification problem (as the perceptron could only do this). The Code regarding the Perceptron class is shown next:

package perceptron;

import java.util.Random;

public class Perceptron {

private double[] weights;

private double threshold;

private double output;

private double error;

private double delta;

public Perceptron(){

this.weights = new double[]{};

this.threshold = 0;

}

public Perceptron(double[] weights, double treshold){

this.weights = weights;

this.threshold = treshold;

}

public void Train(double[][] inputs\_weights, int[] outputs, double threshold, double learning\_rate, int epoch, int selector) // input\_weights are the inputs for the training (2), outputs are the expected outputs, threshold is the defined treshold for the function, learning\_rate is self-explanatory, epoch ys the maximum number of iterations if the error is never 0, selector is used for the type of activation function ( 1 for Step and 2 for Sigmoid)

{

this.threshold = threshold;

int n = inputs\_weights[0].length;

int p = outputs.length;

weights = new double[n];

Random r = new Random();

int i = 0;

int total\_error = 0;

//initialize weights with random values

for(i = 0; i < n; i++) {

weights[i] = r.nextDouble();

}

for(i = 0; i < epoch; i++) {

total\_error = 0;

for(int j = 0; j < p; j++) {

CalculateOutput(inputs\_weights[j], selector);

this.error = outputs[j] - output;

total\_error += this.error;

for(int k = 0; k < n; k++) {

this.delta = learning\_rate \* inputs\_weights[j][k] \* this.error;

weights[k] += this.delta;

}

}

if(total\_error == 0)

break;

}

}

public void CalculateOutput(double[] inputs, int selector){ // 1 for step, 2 for sigmuoid

double sum = 0.0;

for(int i = 0; i < inputs.length; i++) {

sum += weights[i] \* inputs[i];

}

if(selector == 1){

if(sum > threshold)

this.output = 1.0;

else

this.output = 0.0;

} else {

this.output = 1/(1 + Math.exp(this.threshold - sum))

}

}

public double getOutput(){

return this.output;

}

/\*public void CalculateOutputSigmuoid(double[] inputs){

double sum = 0.0;

for(int i = 0; i < inputs.length; i++) {

sum += weights[i] \* inputs[i];

}

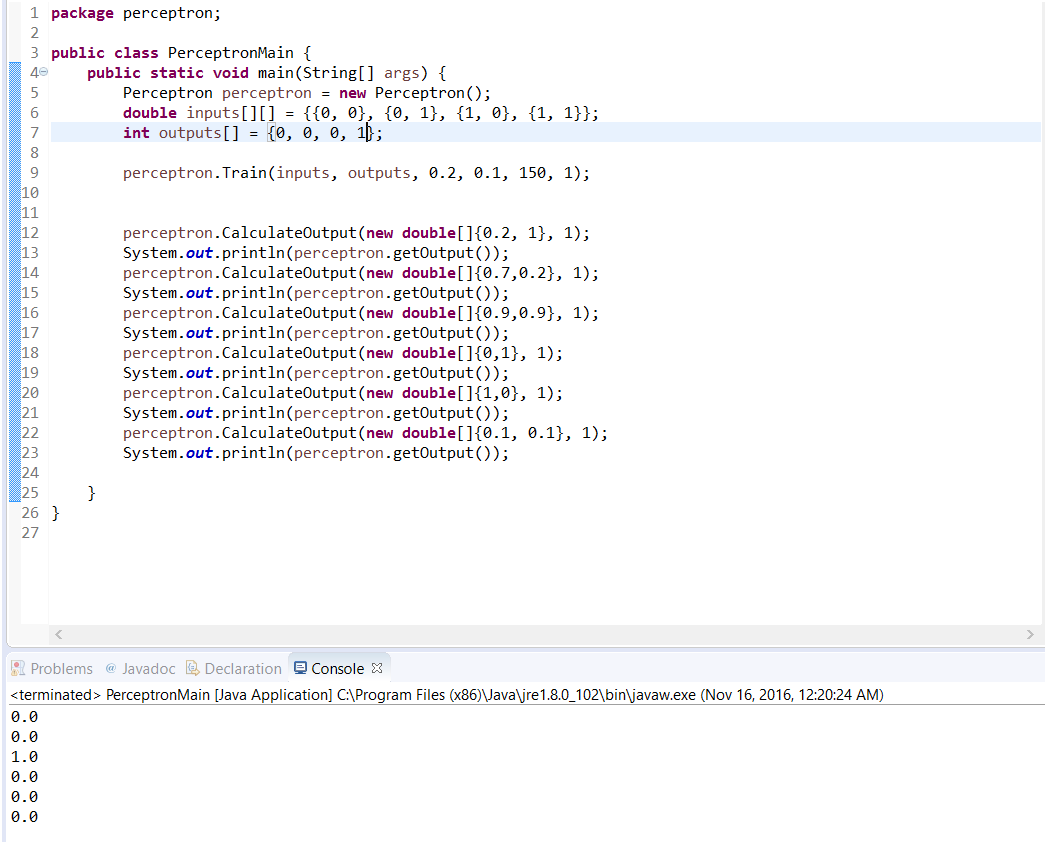
this.output = 2/(1 + Math.exp(this.treshold - sum))

}\*/

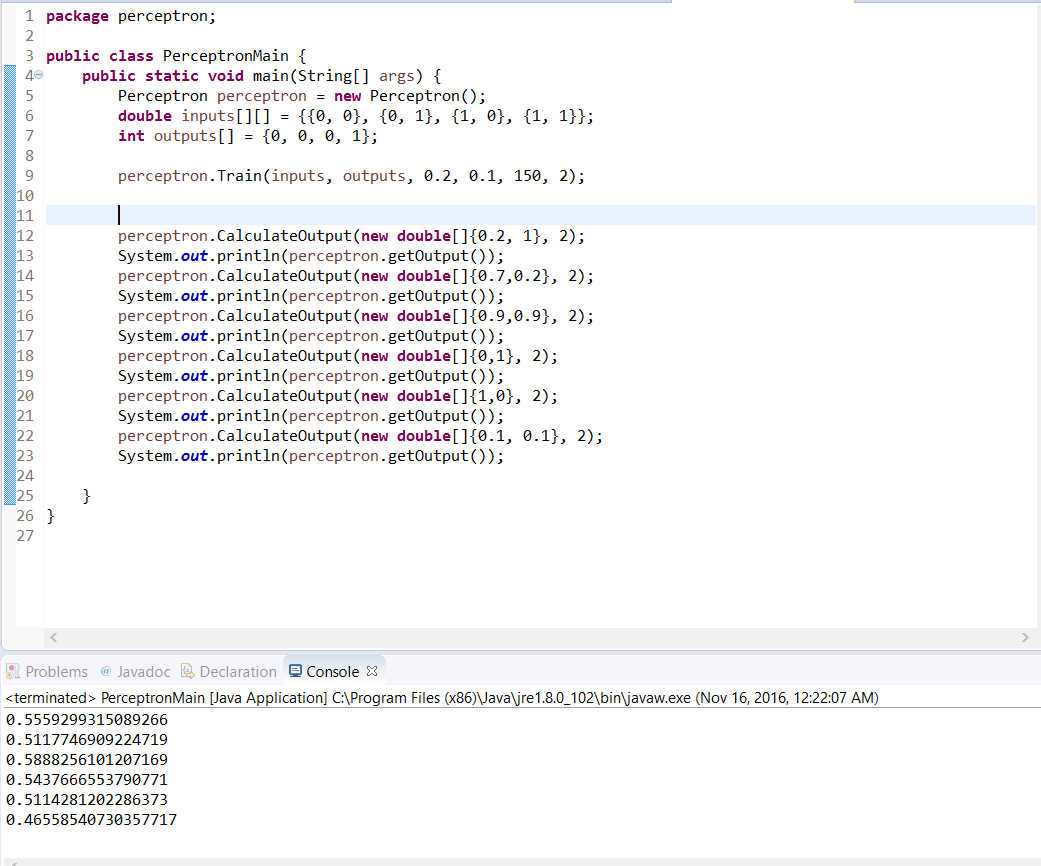
}

The implementation consisted in having a neuron structure with the weights of the links connected to it and also an associated threshold for moving the activation function. The Training function receives the inputs and the expected outputs and starts cycling over these while obtaining the error, if the error is never gone to zero it will stop at a defined limit.

The Activation function was implemented in two ways, a Step function where the outputs are only 0 or 1 (for classification), and a sigmoid function for obtaining values between 0 and 1. The results for training the perceptron for an and with the Step activation function are:

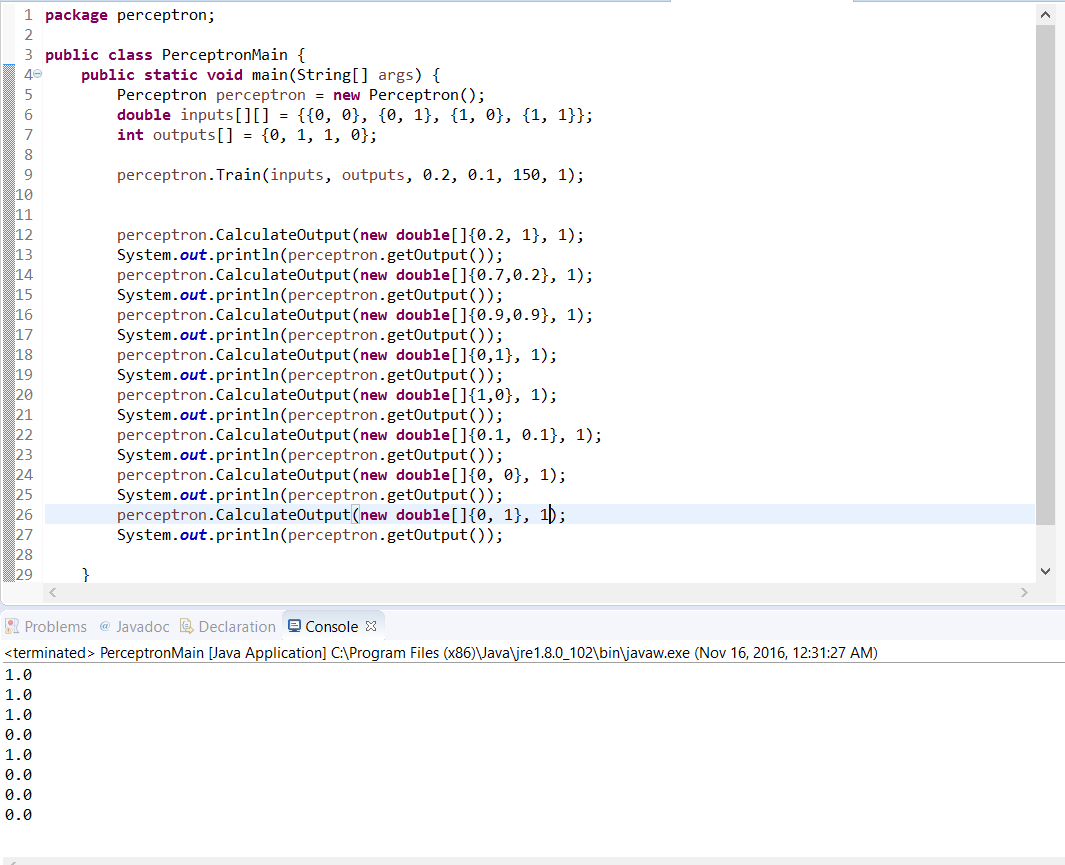


The used threshold is 0.2 and the learning rate is 0.1. Now, for the same values but using the sigmoid function the results are:

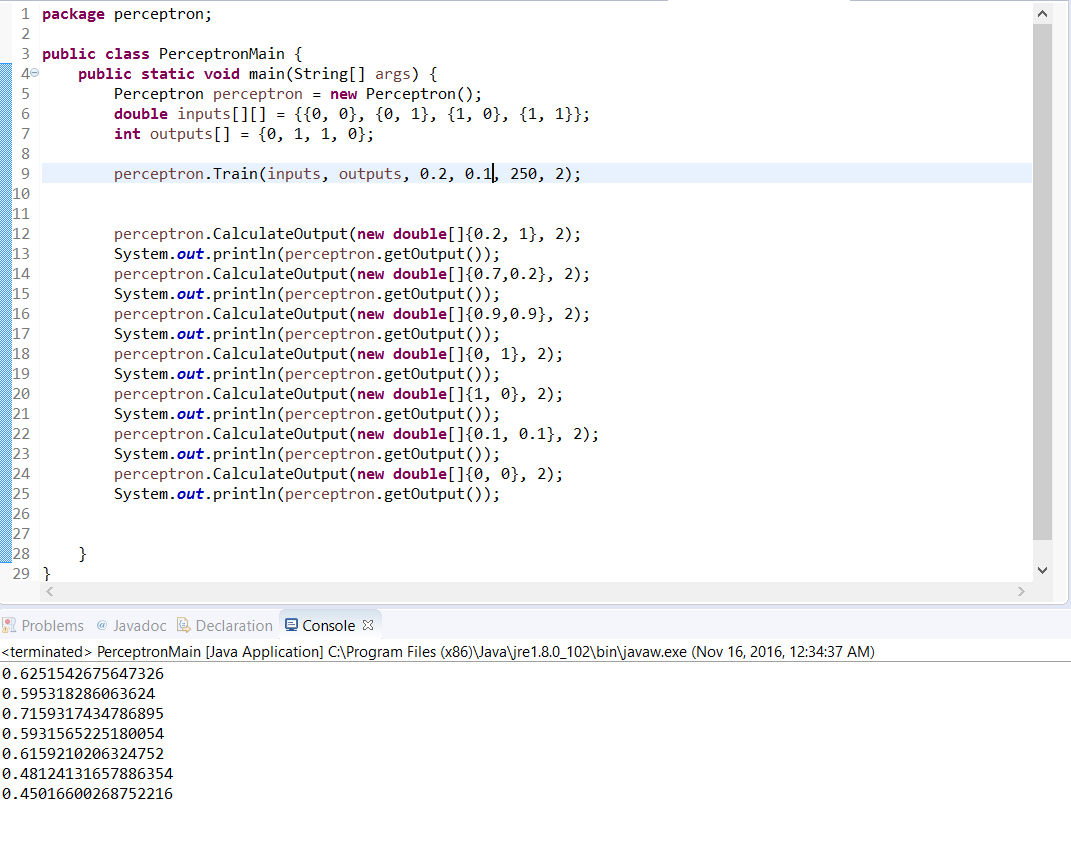


The sigmoid function does not give a classification but a “weight” for the different inputs. For calibrating these results, the threshold and the learning should be moved. In general, for this set the value for the input 0.9, 0.9 should be higher than the rest, as it could be seen.

Now, for trying to classify the XOR gate, the results for the Step activation function are:



As it can be seen, the perceptron couldn't learn the proper way for dividing this boolean equation with a line, it is impossible so the results are unpredictable. With the sigmoid function, the results are:



Here, the answers does not give us enough information, but, the higher value is also for the inputs 0.9, 0.9 so the XOR wasn’t learnt neither.

After analyzing these, it is obvious that the perceptron has capability of classifying objects with a linear separation, for further use it is not capable. The sigmoid function does the same but introduces some type of weight for the inputs so the classification is continuous instead of discrete.