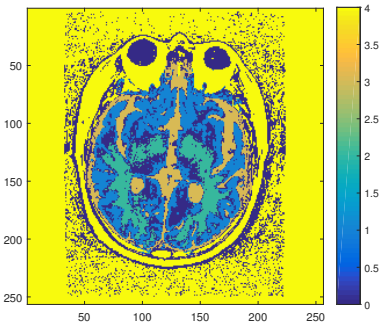
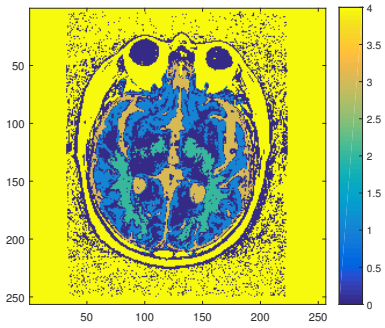
CLP Lab 6 Report

Albert Aparicio Isarn
albert.aparicio.isarn@alu-etsetb.upc.edu
Héctor Esteban Cabezos
hect.esteban@gmail.com

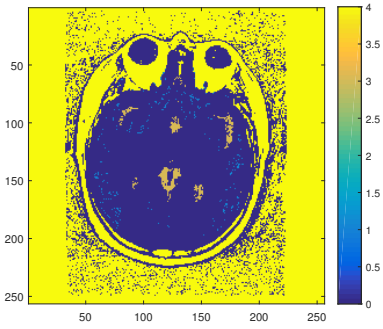
1. Preparación de la base de datos



(a) $P_{min} = 0{,}75$



(b) $P_{min} = 0{,}85$



(c) $P_{min} = 1$

Figura 1: Imagenes resultantes de las labels con varios valores de P_{min}

La elección de P_{min} determina la cantidad de píxeles que se marcan como no pertenecientes a ninguna clase. Estos píxeles se marcan como tales porque su probabilidad es menor a P_{min}.

A medida que se aumenta P_{min}, aumenta el número de estos píxeles, aunque el resto de píxeles son más "fiables", ya que tienen mayor probabilidad.

Por tanto, se debe encontrar un compromiso entre la fiabilidad de la decisión y el número de píxeles no clasificados.

2. Neural networks

2.1. Clasificación con Back-Propagation

Usando este algoritmo, la clasificación da las siguientes probabilidades de error:

$$P_{e_{train}} = 0{,}0190581 \tag{1}$$
$$P_{e_{validation}} = 0{,}0207211 \tag{2}$$
$$P_{e_{test}} = 0{,}0178201 \tag{3}$$

Los resultados de la clasificación se muestran en la figura 2.

2.2. Clasificación con Levenberg-Marquadt

Usando este algoritmo, la clasificación da las siguientes probabilidades de error:

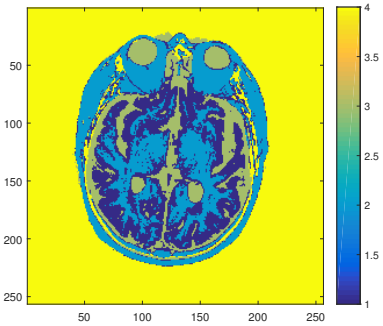
$$P_{e_{train}} = 0{,}00124292 \tag{4}$$
$$P_{e_{validation}} = 0{,}00207211 \tag{5}$$
$$P_{e_{test}} = 0{,}00414422 \tag{6}$$

Los resultados de la clasificación se muestran en la figura 3. En la figura 3c se ve claramente como el pendiente de las curvas de error es mayor, además de que el algoritmo se detiene a las 32 épocas. Claramente la convergencia de este algoritmo es mucho más rápida que la de Back-Propagation.

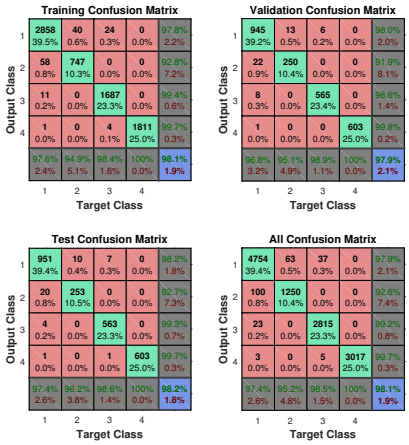
2.3. Comparación de los algoritmos

El algoritmo de Levenberg-Marquadt proporciona una convergencia más rápida, con menor probabilidad de error, que el algoritmo de Back-Propagation.

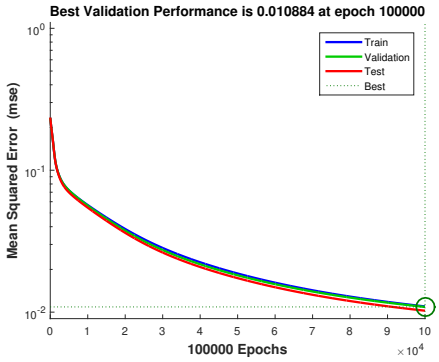
Para este problema se podría decir que el algoritmo de Levenberg-Marquadt es más adecuado.



(a) Resultado de la clasificación

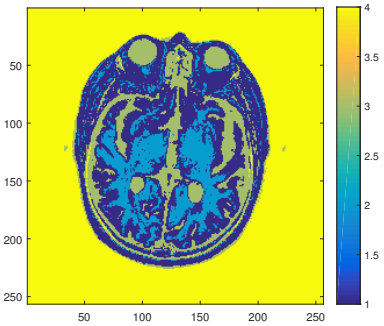


(b) Matriz de confusión

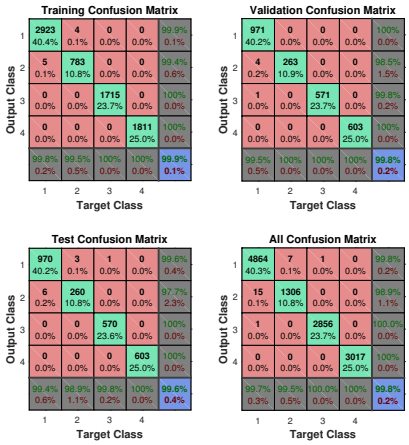


(c) Gráfico de *losses* según cada época

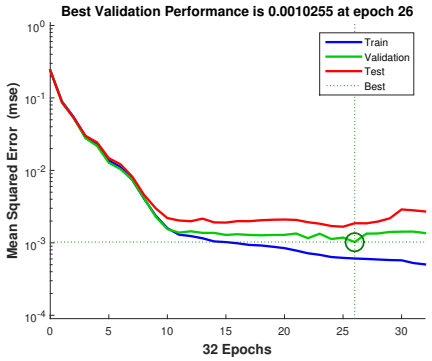
Figura 2: Resultados de la clasificación con Back-Propagation con 100000 *epochs*.



(a) Resultado de la clasificación



(b) Matriz de confusión



(c) Gráfico de *losses* según cada época

Figura 3: Resultados de la clasificación con Levenberg-Marquadt con 32 *epochs*.

2.4. Validación del número de neuronas a utilizar

Según los resultados de las varias clasificaciones, el número óptimo de unidades en la red es de **14**.

Con este tamaño de capa, el error obtenido en test es de $P_{e_{test}} = 0{,}00248653$.

El código usado para esta sección se muestra a continuación:

```
%% Validacion de numero de neuronas en capa oculta
if i_valida_hidden==1;

    MaxLayerSize = 15;
    opt_val_error = zeros(MaxLayerSize, 1);

    % Generate Train, Val & Test BDs and Labels
    % Prepair Train targets
    for hiddenLayerSize = 1:MaxLayerSize
        % Create a Pattern Recognition Network
        net = patternnet(hiddenLayerSize);
        net.performFcn='mse';
        %net.trainFcn='trainscg';  % Conjugate gradient
        %     net.trainFcn='traingd';    %Back Propagation
        net.trainFcn='trainlm';    %Levenberg-Marquadt
        net = configure(net,Brain_Etiq',Target_BD);

        net.divideFcn='divideind'; % The database is divided by indices
        net.divideParam.trainInd=Index_train;
        net.divideParam.valInd=Index_val;
        net.divideParam.testInd=Index_test;
        %         net.divideParam.trainRatio = 1;
        %         net.divideParam.valRatio = 0;
        %         net.divideParam.testRatio = 0;

        net.trainParam.epochs = 100;
        %net.trainParam.max_fail=round(net.trainParam.epochs/10); % Can
    set the
        %number of consecutive high values of the error over epochs in
    the validation set.
        %Used to stop the training.

        net = train(net,Brain_Etiq',Target_BD);% Train the Network

        % Measure val error
        outputs = net(X_val');
        [~, Index_out]=max(outputs);
        opt_val_error(hiddenLayerSize) = length(find(Labels_val~=
    Index_out'))/length(Labels_val);
        %         fprintf(1,' error NN val = %g   \n', opt_val_error(
    hiddenLayerSize));
        %         CM_Val=confusionmat(Labels_val,Index_out);
    end
    % Find optimal layer size
    [~, OptimalLayerSize] = min(opt_val_error);
```

```
    % Create a Pattern Recognition Network from OPTIMAL SIZE
    net = patternnet(hiddenLayerSize);
    net.performFcn='mse';
    %net.trainFcn='trainscg';  % Conjugate gradient
    %     net.trainFcn='traingd';    %Back Propagation
    net.trainFcn='trainlm';    %Levenberg-Marquadt
    net = configure(net,Brain_Etiq',Target_BD);

    net.divideFcn='divideind'; % The database is divided by indices
    net.divideParam.trainInd=Index_train;
    net.divideParam.valInd=Index_val;
    net.divideParam.testInd=Index_test;

    net.trainParam.epochs = 1000;
    %net.trainParam.max_fail=round(net.trainParam.epochs/10); % Can set
    the
    %number of consecutive high values of the error over epochs in the
    validation set.
    %Used to stop the training.

    net = train(net,Brain_Etiq',Target_BD);% Train the Network

    %% Plot train, val and test errors with the number of hidden neurons
    % Measure Train error
    outputs = net(X_train');
    [~, Index_out]=max(outputs);
    NN_Error_train=length(find(Labels_train~=Index_out'))/length(
    Labels_train);
    fprintf(1,' OPTIMAL error NN train = %g   \n', NN_Error_train);
    CM_Train=confusionmat(Labels_train,Index_out)
    % Measure val error
    outputs = net(X_val');
    [~, Index_out]=max(outputs);
    NN_Error_val=length(find(Labels_val~=Index_out'))/length(Labels_val);
    fprintf(1,' OPTIMAL error NN val = %g   \n', NN_Error_val);
    CM_Val=confusionmat(Labels_val,Index_out)
    % Measure Test error
    outputs = net(X_test');
    [~, Index_out]=max(outputs);
    NN_Error_test=length(find(Labels_test~=Index_out'))/length(
    Labels_test);
    fprintf(1,' OPTIMAL error NN test = %g   \n', NN_Error_test);
    CM_Test=confusionmat(Labels_test,Index_out)

end
```

2_NN_val_opt.m

3. Decision trees

La clasificación ha dado como resultado las siguientes probabilidades de error:

$$P_{e_{train}} = 0{,}00234774 \tag{7}$$
$$P_{e_{validation}} = 0{,}00828844 \tag{8}$$
$$P_{e_{test}} = 0{,}00828844 \tag{9}$$

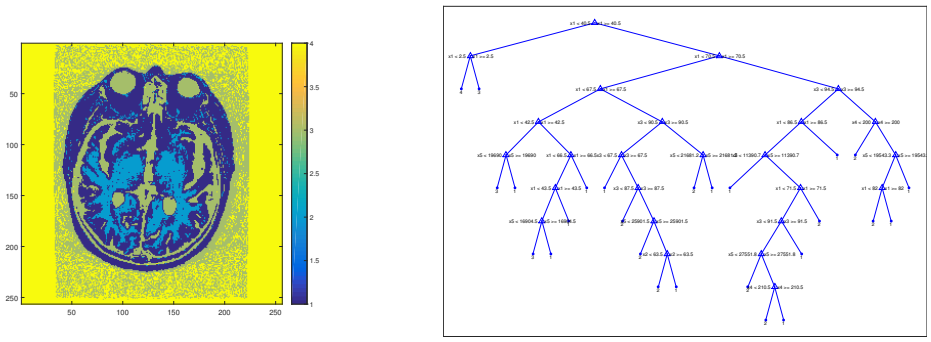
A continuación se muestran las matrices de confusión obtenidas para entrenamiento, validación y test, respectivamente:

$$\begin{pmatrix} 2922 & 6 & 0 & 0 \\ 11 & 776 & 0 & 0 \\ 0 & 0 & 1715 & 0 \\ 0 & 0 & 0 & 1811 \end{pmatrix} \tag{10}$$

$$\begin{pmatrix} 968 & 7 & 1 & 0 \\ 12 & 251 & 0 & 0 \\ 0 & 0 & 571 & 0 \\ 0 & 0 & 0 & 603 \end{pmatrix} \tag{11}$$

$$\begin{pmatrix} 971 & 5 & 0 & 0 \\ 15 & 248 & 0 & 0 \\ 0 & 0 & 571 & 0 \\ 0 & 0 & 0 & 603 \end{pmatrix} \tag{12}$$

El resultado de la clasificación, y el árbol obtenido se muestran en la figura 4.



(a) Resultado de la clasificación (b) Árbol obtenido con el entrenamiento

Figura 4: Resultados de la clasificación con árboles de decisión.

En comparación con el resultado de la red neuronal, el árbol de decisión da un resultado menos preciso y más ruidoso que la red.

3.1. Validación del número de *splits*

El número de *splits* óptimo es **25**. El error de test en este caso es de $P_{e_{test}} = 0{,}00663075$.

El código generado para esta sección se muestra a continuación:

```
%% Validacion de numero MAXIMO DE SPLITS - Tree
if i_valida_split==1;

    % TO DO
    d = 1;
    m = 100;
    val_error = zeros(m-d, 1);
    % Generate Train, Val & Test BDs and Labels
    for MaxNumSplits=d:m
        % Train a tree with the train BD and the train targets
        % Measure Train, Val and Test classification errors

        % Tree classifier design
        tree = fitctree(X_train,Labels_train,'MaxNumSplits',MaxNumSplits)
    ;

        % Measure Val error
        outputs = predict(tree,X_val);
        val_error(MaxNumSplits - d +1)=sum(Labels_val ~= outputs)/length(
    Labels_val);
        %         fprintf('\n-------------------------\n')
        %         fprintf(1,' error Tree val = %g   \n', val_error(
    MaxNumSplits - d +1))
        %         CM_Val=confusionmat(Labels_val,outputs);
    end

    [~, OptimalNumSplits] = min(val_error)

    % Find lowest error and its correspondent MaxNumSplits
    fprintf('\n-------------------------\n')
    fprintf(1,' Optimal Split Size = %g   \n', OptimalNumSplits)

    %% TREE IMAGE CLASSIFICATION
    outputs = predict(tree,Brain_5);
    Aux=reshape(outputs,Ndim,Ndim);
    figure('name','Optimal Tree Classified Image')
    imagesc(Aux)
    axis image
    colorbar

    % Plot train, val and test errors with the number of MaxNumSplits
    % Tree classifier design with optimal number of splits
    tree = fitctree(X_train,Labels_train,'MaxNumSplits',OptimalNumSplits)
    ;
```

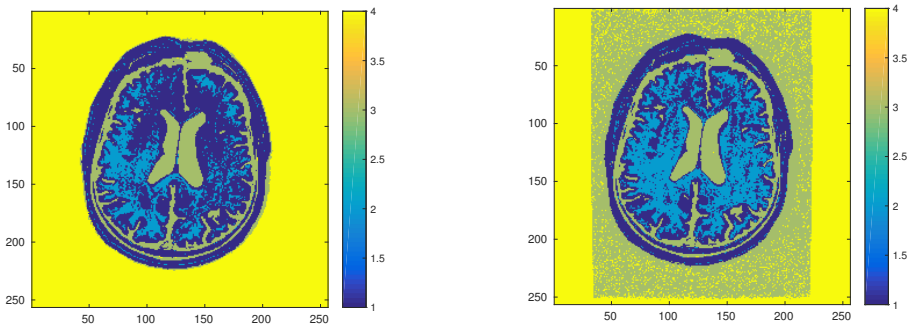
```
    view(tree,'mode','graph');
    view(tree)

    % Measure Train error
    outputs = predict(tree,X_train);
    Tree_Pe_train=sum(Labels_train ~= outputs)/length(Labels_train);
    fprintf('\n------- TREE CLASSIFIER ------------------\n')
    fprintf(1,' error Tree train = %g   \n', Tree_Pe_train)
    CM_Train=confusionmat(Labels_train,outputs)
    % Measure Val error
    outputs = predict(tree,X_val);
    Tree_Pe_val=sum(Labels_val ~= outputs)/length(Labels_val);
    fprintf('\n-------------------------\n')
    fprintf(1,' error Tree val = %g   \n', Tree_Pe_val)
    CM_Val=confusionmat(Labels_val,outputs)
    % Measure Test error
    outputs = predict(tree,X_test);
    Tree_Pe_test=sum(Labels_test ~= outputs)/length(Labels_test);
    fprintf('\n-------------------------\n')
    fprintf(1,' error Tree test = %g   \n', Tree_Pe_test)
    CM_Test=confusionmat(Labels_test,outputs)
    % END TO DO
end
```

3_TREE_val_opt.m

La clasificación es más precisa con redes neuronales, aunque su entrenamiento sea más costoso computacionalmente y lento.

4. Clasificación de una imagen nueva

La clasificación con árboles resulta ruidosa. Con NN, la clasificación es más precisa. Los resultados de la clasificación se muestran en la figura 5.



(a) Clasificación con Red Neuronal (b) Clasificación con Árboles de Decisión

Figura 5: Resultados de la clasificación con Red Neuronal y Árbol de decisión óptimos.