

veronica morante pindado | Aprendizaje automático

HAPT Data Set

Smartphone-Based Recognition of Human Activities and Postural Transitions

Índice

[Introducción 2](#_Toc474609898)

[Elección de técnicas de aprendizaje automático 2](#_Toc474609899)

[Regresión logística regularizada 3](#_Toc474609900)

[Redes neuronales 7](#_Toc474609902)

[Support vector machines 12](#_Toc474609903)

[Análisis de errores 13](#_Toc474609904)

[Comparación resultados 15](#_Toc474609905)

[Apéndice 16](#_Toc474609906)

[Código de regresión logística regularizada 16](#_Toc474609907)

[Codigo de Redes neuronales 17](#_Toc474609909)

# Introducción

En este trabajo se va analizar un dataset en Smartphone sobre el reconocimiento de actividades humanas. Los datos se han obtenido con 30 voluntarios, de cada uno de ellos se han obtenido 17 señales como la aceleración, el giroscopio, etc.…Posteriormente de cada una de estas señales se han obtenido 17 medidas como la media, la varianza, el máximo, el mínimo…Tras segmentar en ventanas da lugar a 561 características y 7767 ejemplos de entrenamiento. En X\_train están cargados estos datos y en y\_train.txt están cargados los datos de cual actividad corresponde, en total hay 12. Tras cargar los datos aplicaremos las técnicas de aprendizaje automático vistas en clase. Para comprobar la efectividad del sistema tenemos unos datos de prueba con 3162 ejemplos que dividiremos por la mitad para hacer ejemplos de validación y de test. Para finalizar comparamos todas las técnicas y explicaré cual es la técnica que se aproxima mejor a este conjunto de datos.

Tenemos las siguientes actividades:

1 WALKING

2 WALKING\_UPSTAIRS

3 WALKING\_DOWNSTAIRS

4 SITTING

5 STANDING

6 LAYING

7 STAND\_TO\_SIT

8 SIT\_TO\_STAND

9 SIT\_TO\_LIE

10 LIE\_TO\_SIT

11 STAND\_TO\_LIE

12 LIE\_TO\_STAND

# Elección de técnicas de aprendizaje automático

Tras revisar las técnicas que hemos dado en clase para clasificación, he decido no aplicar la técnica de regresión lineal debido a que los datos tienes muchas características y son difíciles de separar. He decidido aplicar regresión logística regularizada multiclase, ya que permite dividir con una curva de manera sencilla el conjunto de datos en grupos. También he decidido aplicar redes neuronales de tres capas, esta técnica permite combinar los datos de entrada para producir una salida. Por último utilizaré Support vector machines para calcular el modelo que más se ajusta a los datos.

# Regresión logística regularizada

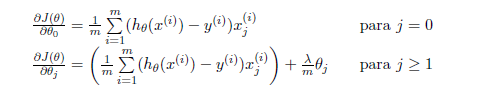
Primero de todo cargamos los datos:

load("X\_train.txt")

load("y\_train.txt")

Posteriormente tenemos que entrenar 12 clasificadores diferentes, uno para cada actividad. Primero crearemos una función para calcular el coste y el gradiente de la regresión logística. Para calcular el coste utilizo la siguiente función: 

y para el gradiente:



El código se puede ver en el [código 2](#codigo2)

A continuación, entrenaré un clasificador para cada una de las 12 clases, crearé una función que devolverá una matriz donde cada fila corresponderá a los parámetros aprendidos para el clasificador de cada una de las clases. Esta función recibe un vector de etiquetas de 1 al 12. Primero obtendremos un vector de etiquetas que indica si el ejemplo de entrenamiento pertenece a la clase o no con 1 u 0. Llamaré a la función de coste y a fmincg que venía con la práctica 3.El código viene en [código3](#codigo3)

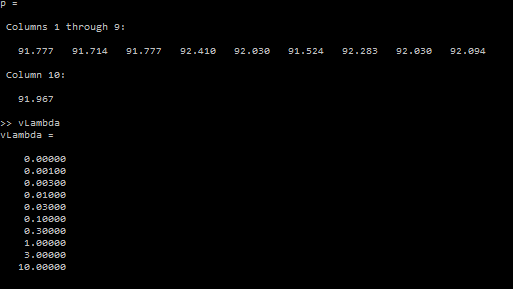
num\_etiquetas = 12;

[ all\_theta ] = oneVsAll (X\_train, y\_train , num\_etiquetas , lambda)

Para calcular la efectividad calculamos el porcentaje de aciertos con la siguiente función: [codigo4](#codigo4).Para cada ejemplo de validación calculamos la probabilidad de que pertenezca a cada una de las clases, nos quedaremos con la etiqueta para la que obtenga el valor máximo.

Creamos un bucle con diferentes valores de Lambda para comprobar cual se ajusta más, lo calculamos con los ejemplos de validación y nos quedaremos con el que más se ajuste:

|  |
| --- |
|  |
| vLambda = [0; 0.001; 0.003; 0.01; 0.03; 0.1; 0.3; 1; 3; 10];  for i = 1:rows(vLambda)  [ all\_theta ] = oneVsAll (X\_train, y\_train , num\_etiquetas , vLambda(i));  [porcentaje]=calcularPorcentaje(X\_val,all\_theta,y\_val);  vLambda(i)  pval(i)=porcentaje  ptrain(i)=calcularPorcentaje(X\_train,all\_theta,y\_train);  endfor |



A continuación, se muestran los valores de lambda y el porcentaje de acierto con los datos de validación:

Tabla RLR-LAMBDA

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| 0 | 0,001 | 0,003 | 0,01 | 0,03 | 0,1 | 0,3 | 1 | 3 | 10 |
| 91,77% | 91,71% | 91,77% | 92,41% | 92,03% | 91,52% | 92,28% | 92,03% | 92,09% | 91,9% |

Ahora dibujaremos la gráfica de porcentaje con los valores de entrenamiento y validación:

|  |
| --- |
| Imagen1-RLR-lambdaImagen 2-RLR-lambda |

Para realizar más pruebas vamos a cambiar el número de iteraciones y utilizar el mejor lambda calculado anteriormente 0,01, los resultados se muestran en la siguiente tabla:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |

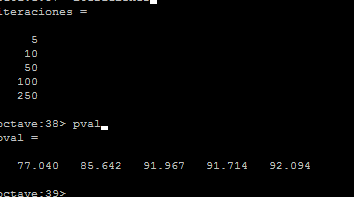
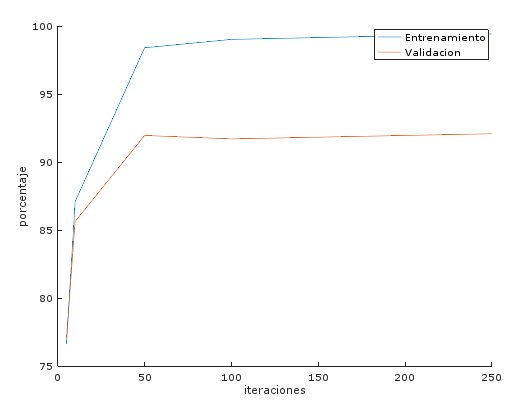


Tabla RLR-Iteraciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 5 | 10 | 50 | 100 | 250 |
| 77,40% | 85,646% | 91,267% | 91,714% | 92,094% |

Imagen 3-RLR-iteraciones

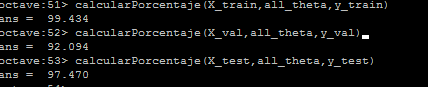
Por último, con los datos de test, comprobaremos el porcentaje de aciertos para verificar que nuestro programa clasifica correctamente y no solo lo ha “aprendido de memoria”:

load("xtest.txt")

load("ytest.txt")

[porcentaje]=calcularPorcentaje(X\_test,all\_theta,y\_test)

En la siguiente imagen se muestra el porcentaje de aciertos con las mejores características lambda=0.01 y 250 iteraciones con los tres conjuntos de datos. Podemos ver que el porcentaje de ejemplos de test es muy alto.



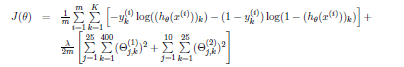
Como conclusión de regresión logística regularizada podemos decir que para nuestro conjunto de datos de más de 7000 ejemplos y 500 características apenas influye lambda para una mayor precisión, todos los porcentajes son bastante parecidos debido a la diversidad de ejemplos y características para diferenciar unos de otros. Lo que si varía es el número de aciertos con el número de iteraciones, cuantas más hay, el porcentaje es mayor, pero a partir de 50 iteraciones la variación es mínima.

# Redes neuronales

La red neural que he creado está formada por tres capas, la primera capa está formada por 561 unidades, la segunda formada por 25(inicialmente, después se modificará) y la última formada por 12. Inicializamos Theta1 y theta2 con pesos aleatorios y los juntamos en inithial\_tetha

|  |
| --- |
|  |
| options = optimset ( 'GradObj' , 'on' , 'MaxIter' , 50);  lambda = 1;  num\_entradas = 561;  num\_ocultas = 25;  num\_etiquetas=12;  Theta1\_inicial=pesosAleatorios(25,562);  Theta2\_inicial=pesosAleatorios(12,26);  initial\_theta = [ Theta1\_inicial(:); Theta2\_inicial(:)]; |

A continuación, calculamos la función de coste para calcular el coste regularizado con la siguiente formula:



Y para calcular el gradiente necesitaremos la derivada de sigmoide [(codigo5](#codigo5)), vamos a utilizar el algoritmo de retro-propagación que consiste en ejecutar una pasada hacia delante, posteriormente una pasada hacia atrás para comprobar el error que se ha producido en la salida. Esto se puede ver en el [código7](#codigo7).

Con la función de coste que he creado ya podemos entrenar la red neuronal con la función fmincg proporcionada en la práctica 4 e inicializando los parámetros con el [codigo8](#codigo8) .

[Theta1,Theta2]=entrenamiento(num\_entradas,num\_etiquetas,num\_ocultas,X\_train,y\_train,initial\_theta)

A continuación, voy a crear un bucle en el que comprobare con diferentes valores de lambda y los ejemplos de validación cual se ajusta más:

|  |
| --- |
|  |
| vLambda = [0; 0.001; 0.003; 0.01; 0.03; 0.1; 0.3; 1; 3; 10];  for i = 1:rows(vLambda)  [Theta1,Theta2]=entrenamiento(num\_entradas,num\_etiquetas,num\_ocultas,X\_train,y\_train,initial\_theta,vLambda(i));  vLambda(i);  pval(i)=calcularPorcentaje(Theta1,Theta2,X\_val,y\_val);  ptrain(i)=calcularPorcentaje(Theta1,Theta2,X\_train,y\_train);  endfor |

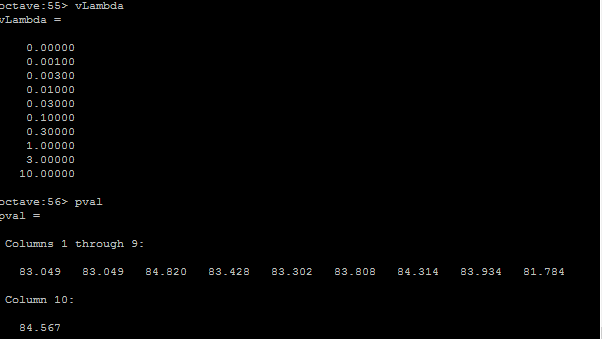
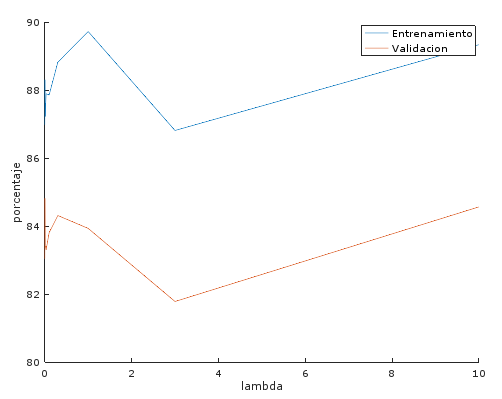
Obtuve los siguientes resultados, el que más se ajusta es 0,003

Tabla RN-Lambda

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| 0 | 0,001 | 0,003 | 0,01 | 0,03 | 0,1 | 0,3 | 1 | 3 | 10 |
| 83,05% | 83,05% | 84,82% | 83,42% | 83,30% | 84,80% | 84,31% | 83,93% | 81,8% | 84,5% |

Pintamos la gráfica con el porcentaje de ejemplos de entrenamiento y de validación:

Imagen 4RN-lambda

Con la mejor lambda obtenido anteriormente (0,03), creamos otro bucle, pero ahora cambiando el número de nodos de la capa oculta:

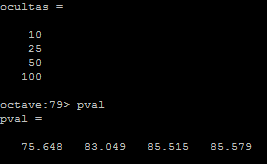
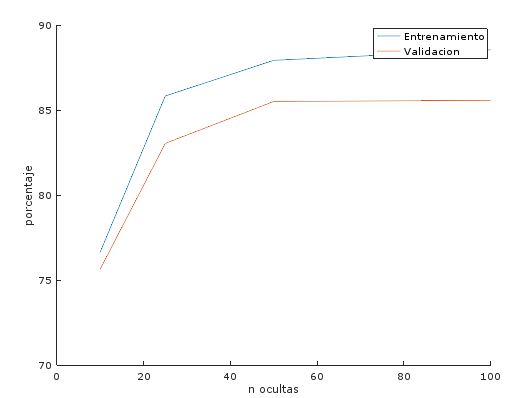


Tabla RN-capa oculta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 10 | 25 | 50 | 100 |
| 75,648% | 83,049% | 85,515% | 85,579% |

Imagen5-RN-ocultas

Para finalizar cambiamos el número de iteraciones con la mejor lambda obtenida y con el mejor número de nodos de la capa oculta obtenidos:

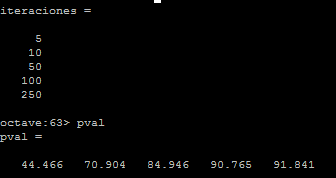
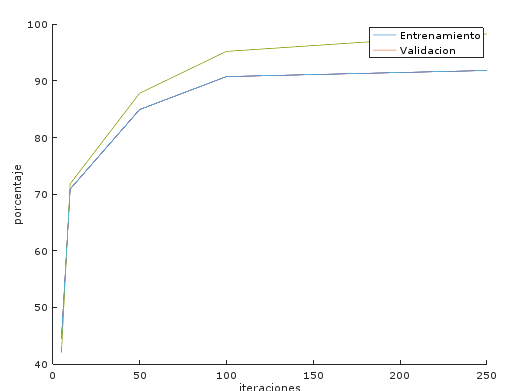
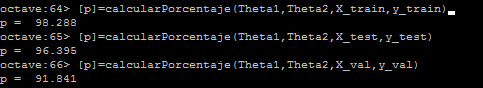


Tabla RN-iteraciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 5 | 10 | 50 | 100 | 250 |
| 44,466% | 70,904% | 84,496% | 90,765% | 91,841% |

Imagen6-RN-iteraciones

Para finalizar comprobamos con los datos de test la efectividad de nuestro sistema, como podemos ver es bastante alta



Como conclusión al igual que en regresión logística lambda apenas influye para una mayor precisión, en cuanto al número de iteraciones también hay un mayor número de efectividad, pero en redes neuronales necesita un número mayor de iteraciones para ajustarse más, en 100 iteraciones se estabiliza el porcentaje con variaciones mínimas. Debido a que nuestros datos de entrenamiento son bastantes y tienen muchas características el número de nodos de la capa oculta necesita ser mayor, para que se combinen los datos de la primera capa.

# Support vector machines

Como la práctica 6 que se ha realizado durante la asignatura solo es para dos clases, he tenido que instalar la biblioteca libsvm que permite calcular el modelo multiclase que más se ajuste con las siguientes ordenes:

model = svmtrain(y\_train,X\_train,'-c 0.3 -h 0');

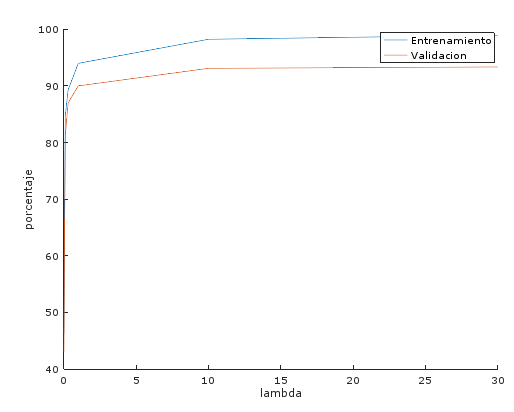
p=svmpredict(y\_train,X\_train,model)

p=svmpredict(y\_val,X\_val, model);

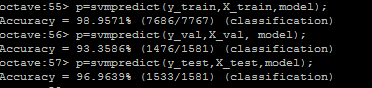
El parámetro C lo he ido variando y comprobando el porcentaje de acierto con los datos de validación. El resultado se puede ver en la siguiente tabla:

Tabla SVM-C

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| c | 0.01 | 0.1 | 0.3 | 1 | 10 | 30 |
| Val | 43,2% | 81,46% | 87,1% | 90,259% | 93,10% | 93,35% |
| train | 46,29% | 84,62% | 89,29% | 93,98% | 98,233% | 98,95% |

Imagen7-SVM-c

Por ultimo como en las dos técnicas anteriores, calcularemos el porcentaje de aciertos con los datos de test con el mejor C (30) y obtendremos los datos para los tres conjuntos de datos.

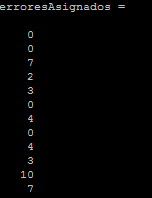
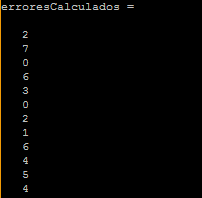


En conclusión, de las SVM, cuanto mayor es el parámetro C, mayor es la efectividad del sistema, esto es debido a que cuanto mayor es este parámetro la SVM es más estricta al momento de permitir errores y los penaliza, por tanto, el sistema es mejor.

# Análisis de errores

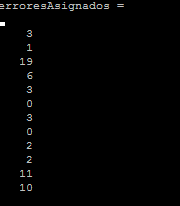
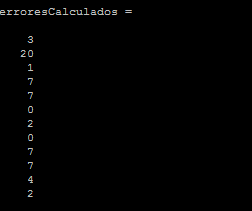
En este apartado voy a comprobar en qué actividades falla más el sistema, para ello he creado dos funciones nuevas [código10](#codigo10) y [codigo11](#codigo11),la primera función devuelve la posición del vector que no ha acertado y la segunda función devuelve cada actividad con el número de fallos.

Primero comprobamos con la regresión logística regularizada con los ejemplos de test, en total devuelve 40 fall0s:

En la primera imagen vemos el número de errores según y\_test y en la segunda imagen vemos lo que nos ha devuelto nuestro sistema. La actividad con la que más se equivoca es la 11. Stand to lie y seguida de 3 WALKING\_DOWNSTAIRS y 12 LIE\_TO\_STAND. Lo que vemos con estos números es que por ejemplos actividades muy parecidas e intermedias como la transición de estar de pie a tumbarse o de tumbarse a estar de pie las confunde. Este problema ocurre con subir y bajar escaleras, Nuestro sistema ha reconocido que era subir escaleras mientras que por el contrario era bajar.

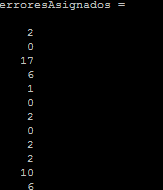
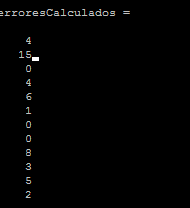
Ahora vamos a hacer las mismas comprobaciones con redes neuronales:

En este caso hay 60 errores, como ocurre con regresión logística se equivoca entre la actividad 11 y 12, pero el mayor número de errores es con subir y bajar escaleras que las confunde.

Por ultimo probare con SVM:

En este caso hay 48 errores,

Igual que los dos metodos anteriores ocurre la actividad 2 y 3 las confunde.Ahora nuestro reconocedor pone que la actividad 9 SIT\_TO\_LIE aparece 8 veces en cambio en solo aparece 2 en verdad la confunde con la actividad 11 y 12 que tambien tiene que cambiar de posicion a tumbarse.

# Comparación resultados

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Logística regularizada  (Lambda=0.01,  Iteraciones=250) | Redes neuronales  (lambda=0,03,  Capa oculta=100,  Iteraciones=250) | SVM  (c=30) |
| Datos de entrenamiento | 99,434% | 98,288% | 98,95% |
| Datos de validación | 92,094% | 91,241% | 93,35% |
| Datos de prueba | 97,470% | 96,395% | 96,96% |

Tras comparar las tres técnicas nos damos cuenta de que las tres funcionan perfectamente, no hay influido apenas el número de características(n) y el número de ejemplos de entrenamiento. Supuestamente con los apuntes tendríamos que suponer que con una n pequeña y m intermedia lo que mejor se ajustaría seria SVM, pero podemos ver con los porcentajes que todas tienen un 98% o 99%.En cuanto a la técnica que más se ha ajustado seria regresión logística regularizada, apenas con 50 iteraciones el sistema ya llegaba a un 90% de efectividad al contrario de redes neuronales que solo llegaba al 80%,esta técnica ha tenido que tener el doble de iteraciones para llegar al 90%.SVM también ha tenido un porcentaje bastante alto a medida que aumentaba el parámetro C, al contrario de las dos técnicas anteriores que han tenido resultado similares respecto a la variación de lambda. En lo que se refiere al tiempo la mejor ha sido SVM, ha tardado mucho menos tiempo que las otras dos técnicas.

# Apéndice

A continuación, escribo el código que he utilizado, algunas funciones que están en varios algoritmos no las repito.

## Código de regresión logística regularizada

|  |
| --- |
| Codigo 1: |
| function [g] = sigmoide (z)  g = 1 ./ (1 + exp (-z));  endfunction |

|  |
| --- |
| Codigo2: |
| function[J,grad] =lrCostFunction (theta,X,y,lambda)  g = sigmoide(theta'\*X');  g = g';  m = rows(X);  grad = 1 / m .\*sum((g.-y).\*X.+lambda/m.\*theta');  grad = grad';  sumaj = sum(-y .\* log(g) - (1 - y) .\* log(1 - g) .+ lambda/(2\*m)\*sum((theta.^2)));  J = (1/m)\*sumaj;  endfunction |

|  |
| --- |
| Codigo 3: |
| function [ all\_theta ] = oneVsAll (X, y , num\_etiquetas , lambda)  options = optimset ( 'GradObj' , 'on' , 'MaxIter' , 50);  vector = ones(rows(X),1);  X=[vector X];  z = (y(1:rows(X), 1) == 1:num\_etiquetas);  for cont = 1:num\_etiquetas  initial\_theta = zeros (columns(X) , 1);  theta = fmincg (@( t ) (lrCostFunction (t , X, z(:,cont), lambda)), initial\_theta, options);  all\_theta(:,cont) = theta;  endfor endfunction |

|  |
| --- |
| Codigo 4: |
| function[porcentaje] =calcularPorcentaje(X\_train,all\_theta,y\_train)  vector = ones(rows(X\_train),1);  X\_train=[vector X\_train];  c = sigmoide(all\_theta'\*X\_train');  [M,i] = max(c);  i = i';  aciertos = find(y\_train== i);  porcentaje = rows(aciertos) / rows(y\_train) \* 100;  endfunction |

## Codigo de Redes neuronales

|  |
| --- |
| Codigo5: |
| function [g] = derivadaSigmoide (z)  g = sigmoide(z) .\* (1-sigmoide(z));  endfunction |

|  |
| --- |
| Codigo 6: |
| function W= pesosAleatorios (L\_in , L\_out)  epsilon=sqrt(6)/sqrt(L\_in+L\_out);  W = rand(L\_in,L\_out)\*(2\*epsilon)-epsilon;  endfunction |

|  |
| --- |
| Codigo7: |
| function [ J grad ] = costeRN(params\_rn , num\_entradas , num\_ocultas ,num\_etiquetas  , X,y ,lambda )  Theta1 = reshape (params\_rn (1: num\_ocultas \* ( num\_entradas + 1) )  ,num\_ocultas , ( num\_entradas + 1) ) ;  Theta2 = reshape (params\_rn ((1 + (num\_ocultas \* ( num\_entradas + 1) ) ) :  end ) , num\_etiquetas,( num\_ocultas+ 1) ) ;  J = 0;  grad1 = zeros(size(Theta1));  grad2 = zeros(size(Theta2));  vector = ones(rows(X),1);  X=[vector X];  m=size(X,1);  z = (y(1:rows(X), 1) == 1:num\_etiquetas);  y1=z;  a1=X;  z2 = a1 \* Theta1';  a2 = sigmoide(z2);  vector2 = ones(rows(z2),1);  z2=[vector2 z2];  a2=[vector2 a2];  z3=a2\*Theta2';  g=a3=sigmoide(z3);  sumaj = sum(sum((-y1) .\* log(g)- (1-y1) .\* log(1 - g)));  sR = (1/m)\*sumaj;  J = sR.+ (lambda/(2\*m)\*((sum(sum(Theta1(:, 2:end).^2)))+ sum(sum(Theta2(:,  2:end).^2)))) ;  sigma3=a3-y1;  sigma2=(sigma3\*Theta2.\*derivadaSigmoide(z2))(:, 2:end);  delta1 = sigma2'\*a1;  delta2 = sigma3'\*a2;  grad1 = delta1./m + (lambda/m)\*[zeros(size(Theta1,1), 1) Theta1(:, 2:end)];  grad2 = delta2./m + (lambda/m)\*[zeros(size(Theta2,1), 1) Theta2(:, 2:end)];  grad=[grad1(:);grad2(:)];  endfunction |

|  |
| --- |
| Codigo 8: |
| function  [Theta1,Theta2]=entrenamiento(num\_entradas,num\_etiquetas,num\_ocultas,X\_train,y\_train, initial\_theta)  options = optimset ( 'GradObj' , 'on' , 'MaxIter' , 50);  lambda = 1;  params\_rn= fmincg (@( t ) (costeRN (t , num\_entradas , num\_ocultas ,num\_etiquetas , X\_train, y\_train,  lambda)), initial\_theta, options);  Theta1 = reshape (params\_rn (1: num\_ocultas \* ( num\_entradas + 1) ) ,num\_ocultas , (  num\_entradas + 1) ) ;  Theta2 = reshape (params\_rn ((1 + (num\_ocultas \* ( num\_entradas + 1) ) ) : end ) ,  num\_etiquetas,( num\_ocultas+ 1) ) ;  endfunction |

|  |
| --- |
| Codigo9: |
| function [p]=calcularPorcentaje(Theta1,Theta2,X,y)  vector = ones(rows(X),1);  X=[vector X];  a2 = sigmoide(Theta1\*X');  a2=a2';  a2=[vector a2];  h=sigmoide(Theta2\*a2');  [M,i] = max(h);  i = i';  aciertos = find(y == i);  p= rows(aciertos) / rows(y) \* 100;  endfunction |

|  |
| --- |
| Codigo10: |
| function[porcentaje,aciertos,i] =errores(X\_train,all\_theta,y\_train)  vector = ones(rows(X\_train),1);  X\_train=[vector X\_train];  c = sigmoide(all\_theta'\*X\_train');  [M,i] = max(c);  i = i';  aciertos = find(y\_train!= i);  porcentaje = rows(aciertos) / rows(y\_train) \* 100;  endfunction |

|  |
| --- |
| Codigo11: |
| function [erroresAsignados,erroresCalculados] = calcularErrores (error, y\_test , num\_etiquetas,ycalculada)  erroresAsignados= zeros(num\_etiquetas, 1);  erroresCalculados= zeros(num\_etiquetas, 1);  for i=1:rows(error)  erroresAsignados(y\_test(error(i))) = erroresAsignados(y\_test(error(i))) + 1;  erroresCalculados(ycalculada(error(i))) = erroresCalculados(ycalculada(error(i))) + 1;  endfor  endfunction |