Práctica 3. Redes neuronales de funciones de base radial

Introducción a los modelos computacionales

Ingeniería Informática (Rama Computación)

Escuela Politécnica Superior

Universidad de Córdoba

Curso 2016 – 2017

Autor: Rubén Medina Zamorano - 46068656R

Email: *i32mezar@uco.es*

ÍNDICE

1. INTRODUCCIÓN 3

2. DESCRIPCIÓN DE LOS MODELOS DE REDES NEURONALES 4
2.1 Arquitectura 4
2.2 Organización de capas 4

3. DESCRIPCIÓN DEL PSEUDOCÓDIGO DEL ALGORITMO DE RETROPROPAGACIÓN 6

4. EXPERIMENTOS Y ANÁLISIS DE RESULTADOS 8
4.1. Bases de datos utilizadas 8
4.2. Parámetros de entrada 8
4.3. Resultados obtenidos 9

5. REFERENCIAS BIBLIOGRÁFICAS 15

1. INTRODUCCIÓN

Se trata de la tercera práctica para la asignatura Introducción a los Modelos Computacionales. En ella, procederemos a elaborar una memoria que describa el problema a tratar, analice el pseudocódigo y el funcionamiento del algoritmo utilizado y por último, obtendremos unos resultados que estudiaremos y compararemos para llegar a unas conclusiones.

2. DESCRIPCIÓN DE LOS MODELOS DE REDES NEURONALES

2.1. Arquitectura

El modelo que se desarrollará serán redes neuronales con funciones de base radial. Esto quiere decir, que nuestro modelo tendrá unas entradas y mediante las neuronas internas con función RBF se obtendrán unas salidas. Es similar a ambas prácticas anteriores, con la peculiaridad de utilizar un algoritmo de Kmeans para entrenar y tener funciones RBF en las neuronas de capa oculta.

. Las entradas serán suministradas mediante un fichero de texto, aunque también se podría implementar para introducir los datos manualmente. Y posteriormente, se procesan unas salidas que pueden representar la probabilidad de pertenencia a una clase para clasificación o un valor estimado para regresión, según se índice que en parámetro de entrada.

El funcionamiento de entrenamiento para **clasificación** consiste en seleccionar tantos patrones aleatorios como neuronas queramos en nuestro modelo. Estos patrones serán preferiblemente uno de cada clase, y representarán los centroides iniciales a partir de los cuales se entrenara mediante el algoritmo de Kmedias. Una vez clasificados todo el resto de patrones y reorganizar los centroides, se calculan los radios entre clusters y las distancias de los patrones. Y finalmente, utilizaremos regresión logística para las salidas, con ello pretendemos calcular el error cuadrático medio y la precisión de clasificación.

El funcionamiento de entrenamiento para **regresión** consiste en seleccionar los patrones de forma aleatorio, ya que no tendremos clases. Estos patrones aleatorios serán los centroides, y a partir de ellos aplicaremos el algoritmo Kmedias. Posteriormente, se continuará igualmente con las distancias, terminando con la pseudo inversa de Moore-Penrose. Realizando el producto matricial de esta matriz con las salidas obtendremos los coeficientes Betas para las salidas de regresión.

2.2. Organización de capas

La estructura de nuestro modelo se organizará generalmente con una capa de n entradas, una capa oculta con 10 rbf por defecto y con un único valor de salida en regresión o varias probabilidades en clasificación.

El número de entradas se establecerán mediante el fichero de texto, al igual que las salidas obtenidas. Y el número de capas ocultas o el número de neuronas se establecerán mediante los parámetros de entrada de manera num_rbf es el número de funciones rbf en capa oculta. Además, se incluye sesgo por defecto, y se podrá optar por una regularización en los pesos mediante el parámetro L2.

Los pesos se establecen mediante una matriz de distancias de cada patrón a los centros de las RBF y los coeficientes de la capa oculta se obtiene realizando la matriz pseudoinversa para regresión o aplicando regresión logística para regresión.

3. DESCRIPCIÓN DEL PSEUDOCÓDIGO DEL ALGORITMO DE RETROPROPAGACIÓN

Se detallará el contenido en forma de pseudocódigo de todas las funciones utilizadas en el script desarrollado para el algoritmo.

Función para leer los datos desde los ficheros .csv

```
def lectura_datos(fichero_train, fichero_test)
      file <- pd.read_csv(fichero_train, header=None)
      file2 <- pd.read_csv(fichero_test, header=None)
      train_inputs <- file.values[:, 0:-1]
      train_outputs <- file.values[:, -1:]
      test_inputs <- file2.values[:, 0:-1]
      test_outputs <- file2.values[:, -1:]
Devolver train_inputs, train_outputs, test_inputs, test_outputs
```

Función que inicializa los centroides para n = num_rbfs y escoge 1 patron aleatorio de forma que se coge un patron por cada clase siendo el total num_rbf/num_clases

```
def inicializar_centroides_clas(train_inputs, train_outputs, num_rbf):
      centroides <- matriz_zeros(num_rbf * num_entradas))
      Para i desde 0 hasta num_rbf
            mod <- i mod num_clases;
            Mientras sea verdad
                  index <- patron_random
                  clase <- train_outputs[index]
                  si clase es igual mod
                        salir
            centroides[i]<- train_inputs[index]
      Devuelve centroides
```

Función que aplica el algoritmo de Kmeans según se trate de clasificación o regresión y devuelve los centros y la matriz de distancias de cada patron a los centros.

```
def clustering(clasificacion, train_inputs, train_outputs, num_rbf):
Si es clasificacion
      Centroides<-inicializar_centroides_clas()
      kmedias = KMeans(init=centroides, max_iter=500)
sino
      kmedias = KMeans(init='random', max_iter=500)

kmedias.fit(train_inputs,train_outputs)
centros <- kmedias.cluster_centers_
distancias <- matriz_ceros (train_inputs.shape * num_rbf)
Para i desde 0 hasta num_patrones
      Para j desde 0 hasta num_centros
            distancias[i][j]<-distance.euclidean(train_inputs[i,:],
      centros[j,:])

Devuelve kmedias, distancias, centros
```

Calcula los num_rbf radios de las funciones rbf.

```
def calcular_radios(centros, num_rbf):
      radios <- vector_ceros(num_rbf)
      Para i desde 0 hasta num_rbf
            Para j desde 0 hasta num_rbf
            aux+<-distancia(centros[i,:], centros[j, :])
            radios[i]<-(1.0/(2.0*(num_rbf-1.0))) * aux
Devuelve radios
```

Calcula de la matriz R que se utilizará para la pseudo-inversa de Moore-Penrose en regresión o regresión logística en clasificación.

```
def calcular_matriz_r(distancias, radios):
      matriz_r<-matriz_zeros(num_patrones* centros)
      dist <- elevar_cuadrado (distancias)
      rad <- elevar_cuadrado (radios)
      aux <- exponente (-dist/(2.0*rad))
      matriz_r[:, :-1] <- aux
      matriz_r[:, -1:] <- 1.0 //Fila de sesgo
Devolver matriz_r
```

Calcula la matriz pseudo inversa de Moore- Penrose para regresión y devuelve los coeficientes de los nodos ocultos.

```
def invertir_matriz_regresion(matriz_r, train_outputs):
      #Realiza la pseudoinversa de Moore-Penrose
      pseudo <- pseudo_inv(matriz_r)
      #Realiza el productor matricial de la pseudo y las salidas
deseadas
      coeficientes <- prod_matricial(pseudo, train_outputs)
Devuelve coeficientes
```

Aplica la regresión logística a la matriz R y obtiene una salida en función del parámetro l2 y eta.

```
def logreg_clasificacion(matriz_r, train_outputs, eta, l2):
Si l2
      Logreg<-LogisticRegression('l2',C=1.0/eta).fit(output)
sino
      logreg<-LogisticRegression('l1',C=1.0/eta).fit(output)
 Devuelve logreg
```

4. EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

4.1. Bases de datos utilizadas

Las bases de datos que utilizaremos serán las suministradas en la carpeta /csv, y se dividen en 2 conjuntos, todos con ficheros de train y test:

Para regresión:

- **Función seno**: esta base de datos esta compuesta por 120 patrones de entrenamiento y 41 patrones de test. Ha sido obtenido añadiendo cierto ruido aleatorio a la función seno.
- **Base de datos forest:** esta base de datos esta compuesta por 387 patrones de entrenamiento y 130 patrones de test. Contiene, como entradas o variables independientes, una serie de datos meteorológicos y otras variables descriptoras sobre incendios en bosques al norte de Portugal, y, como salida o variable dependiente, el área quemada.
- **Base de datos CPU**: esta base de datos esta compuesta por 109 patrones de entrenamiento y 100 patrones de test. Contiene como entradas o variables independientes las características de un conjunto de procesadores, y como salida o variable dependiente el rendimiento relativo del procesador.

Para clasificación:

- **Base de datos iris**: tenemos 112 patrones de entrenamiento y 38 para test. Esta base de datos posee 4 variables independientes de entrada, que especifican longitud y ancho de sépalo y pétalo, con 3 clases de salida según corresponda a la clasificación de la flor iris: setosa, virginica o versicolor. Por ejemplo, las salidas serían 0 o 1 perteneciendo a la clase versicolor.
- **Base de datos digits**: está formada por 1275 patrones entrenamiento y 319 patrones de test. La base de datos está formada por un conjunto de dígitos del 0 al 9 escritos en una rejilla de 16x16 píxeles, estos píxeles son las entradas independientes que dan lugar a 10 salidas para clasificar el número correspondiente.

4.2. Parámetros de entrada

A la hora de ejecutar nuestro programa, tiene una serie de parámetros de entrada para modificar el aprendizaje o la estructura de nuestra red. Estos parámetros se modificarán a nivel de código como los ficheros de test y train o el num_rbf. Todos ellos se aplicarán en la función entrenar_rbf:

- **fichero_train**: Fichero csv con los patrones de entrada y salida en entrenamiento.
- **fichero_test**: Fichero csv con los patrones de entrada y salida en test.
- **num_rbf**: Se indican el número de neuronas que tendrá el modelo, al igual que el número de clusters que se formaran.
- **clasificación**: Variable booleana que indicara si utilizaremos el algoritmo para clasificación (True) o regresión (False).
- **eta**: Tasa de aprendizaje para el entrenamiento, que actuará modificando el parámetro C=1/eta.

- **l2**: Representa la regularización de la regresión logística. Si l2 esta True el modelo tiende a obtener valores más pequeños, y si l2 es False (l1) se tienden a obtener valores próximos a 0 y pequeños en valor absoluto (también negativos)

4.3. Resultados obtenidos

Para las bases de datos dadas y los diferentes parámetros realizaremos una serie de experimentos modificando esos parámetros, y ajustando una mejor estructura para minimizar el error en test y obtener un buen modelo que clasifique maximizando la precisión.

En primer lugar, para **regresión** vamos a considerar varios casos de 5%, 10% 25% y 50% del número de patrones para asignar al número de nodos en capa oculta, num_rbf. Y estudiaremos el mejor con un eta= 10e-5, aunque en regresión no se aplica el parámetro eta ni la regularización.

Resultados función seno

Tenemos 120 patrones de entrenamiento, de los cuales 5%=6 , 10%=12, 25%= 30, 50%=60. A partir de estos datos del número de neuronas, obtendremos los valores del error cuadrático para train y test, además el parámetro seguido del símbolo (/) se trata de la desviación típica.

Error cuadrático medio (Media/DT)

Num_rbf	6		12		30		60	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	0.01/0	0.22/0	0.01/0	0.03/0	0.01/0	2.29/1.31	0.01/0	3.51/3.36

En este primer caso, el valor óptimo sería un 10%=12rbf, ya que si tenemos más nodos en capa oculta se mejora en el entrenamiento de forma mínima, pero se sobreentrena y no es bueno para generalizar. Tendría un modelo complejo difícil de interpretar, e interesa mejor el modelo simple de 12 nodos.

Resultados Forest

Tenemos 387 patrones de entrenamiento, de los cuales 5%=19, 10%=39 , 25%=97, 50%= 194. A partir de estos datos del número de neuronas, obtendremos los valores del error cuadrático para train y test, además el parámetro seguido del símbolo (/) se trata de la desviación típica.

Error cuadrático medio (Media/DT)

Num_rbf	19		39		97		194	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	0.001/0	0.0089/0	0.001/0	0.0089/0	0/0	0.009/0	0/0	0.01/0

En este caso ocurre igual que en el anterior, escogeremos el modelo para 5%=19 rbf, ya que produce resultados similares. Sin duda, al coger un árbol con más nodos, es mas complejo y mejora en train, pero no es tan útil para test.

Resultados CPU

Tenemos 109 patrones de entrenamiento, de los cuales 5%=5, 10%=11, 25%=27, 50%=55. A partir de estos datos del número de neuronas, obtendremos los valores del error cuadrático para train y test, además el parámetro seguido del símbolo (/) se trata de la desviación típica.

Error cuadrático medio (Media/DT)

Num_rbf	5		11		27		55	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	0.005/0	0.006/0	0.003/0	0.004/0	0.001/0	0.003/0	0/0	0.002/0

En este caso, también tenemos errores similares para las distintas opciones. Pero por ejemplo podemos optar por el 25% ya que se trata de menos patrones y se puede permitir 27 nodos por ejemplo.

En segundo lugar, para **clasificación** vamos a proceder igualmente para el número de clusters. Y estudiaremos el mejor con un eta= 10e-5 y con la regularización L1. Pero posteriormente, variaremos el valor de eta con ambas regularizaciones L1 y L2. Además, sacaremos la matriz de confusión y probaremos como funciona un problema de clasificación con regresión.

Resultados Iris

Tenemos 112 patrones de entrenamiento, de los cuales 5%=6, 10%=11, 25%=28, 50%=56. A partir de estos datos del número de neuronas, obtendremos los valores del error cuadrático y precisión con media/desviación_típica.

Error cuadrático medio (Media/DT)

Num_rbf	6		11		28		56	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	0.035/0	0.036/ 0.01	0.01/ 0.003	0.05/ 0.01	0/0	0.1/0	0/0	0.1/0

CCR % Precisión (Media/DT)

Num_rbf	6		11		28		56	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	96.4/0	96.3/1.28	98/9,35	94.7/1.66	100/0	89.47/0	100/0	89.6/0

Una vez obtenido el mejor valor de clusters, procederemos a comparar los resultados para los distintos valores del parámetro eta. Y escogeremos el mejor para posteriormente comparar ambos tipos de regularización L2 y L1. En este caso, escogemos el valor **num_rbf=6** ya que se minimiza el error y se maximiza el CCR.

Con regularización L1

	Error cuadrático medio				Precisión CCR %			
	Train		Test		Train		Test	
Eta	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**
0	0.03	0.004	0.031	0.01	96.79	0.43	96.84	1.05
0.1	0.03	0	0.02	0.04	96.43	0	97.37	4.07
0.01	0.03	0.004	0.04	0.04	96.79	0.43	95.26	4.21
0.001	0.03	0.004	0.02	0	96.79	0.43	97.37	0
0.0001	0.03	0.004	0.04	0.03	"	"	95.26	3.06
10e-5	"	"	"	"	"	"	"	"
10e-10	"	"	"	0.01	"	"	95.79	1.28

Con regularización L2

	Error cuadrático medio				Precisión CCR %			
	Train		Test		Train		Test	
Eta	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**
0	0.03	0.004	0.02	0	96.79	0.43	97.37	0
0.1	0.05	0.1	0.03	0.01	94.29	1.07	96.84	1.96
0.01	0.03	0.006	0.005	0.01	96.61	0.66	99.47	1.05
0.001	0.03	0.004	0.06	0.04	96.07	0.43	93.68	4.27
0.0001	0.03	0.003	0.031	0.01	96.61	0.35	96.84	1.05
10e-5	"	"	0.02	0	"	"	97.37	0
10e-10	"	0	"	"	"	0	"	"

Aparentemente se puede considerar que la regularización L2 obtiene mejores resultados que L1. Esto podría explicarse ya que la L1 poda la red poniendo pesos nulos. Por tanto, el caso que escogeremos será el eta=0.01 para L2 que obtiene casi el 100% de clasificación.

En cuanto a los coeficientes para L2 y para L1 se diferencian en lo siguiente.

Coeficientes para L1

```
[[  5.59530956   0.         -8.91176692   4.88670251 -11.60412099   0.
    0.        ]
 [ -4.0581744   20.65528871  -1.44661853   0.          6.58250375
  -18.64398765  -2.30869265]
 [-64.51290921 -12.58724511  -0.24924742   0.          0.          12.61194858
   4.41242739]]
Numero clases  3
Numero nodos ocultos  7
```

Tenemos 3 clases de salida y 7 nodos en capa oculta (6 rbf y sesgo). Pero si nos fijamos en los pesos tienen pesos positivos, negativos y otros a 0 que son las ramas que poda la regularización L1.

Sin embargo, en la regularización L2 no se podan nodos como podemos ver. Solo minimiza el valor pero sin anularlo.

Coeficientes para L2

```
[[  4.31310714  -2.25722668  -2.64178268   5.4421138   -5.97747997
   -4.19990121  -1.27376923]
 [ -1.76670334  14.11106978 -12.71417657  -1.3433299    6.6315462
   -1.0308392   -2.68353064]
 [ -8.28156477 -12.24636426  10.12219661  -7.86874594  -1.16886851
    2.98440474   1.35646246]]
Numero clases  3
Numero nodos ocultos  7
```

Y por último, mostramos la matriz de confusión en la que podemos ver que apenas hay errores en la clasificación de los patrones.

```
[[13  0  0]
 [ 0 10  2]
 [ 0  0 13]]
```

Solo existen dos patrones mal clasificados, y en otros casos con distinta semilla clasifica al 100%.

Al igual que con clasificación, podemos aplicar este problema pero desde el punto de vista de regresión. Con los parámetros L2 y eta=0.01 obtenemos los siguientes resultados. Que son bastante buenos al tratarse de un problema de clasificación.

```
INFORME FINAL
------------------------
Error de entrenamiento (Media +- DT): 0.061761 +- 0.015544
Error de test (Media +- DT): 0.048167 +- 0.008820
CCR de entrenamiento (Media +- DT): 0.00% +- 0.000000%
CCR de test (Media +- DT): 0.00% +- 0.000000%
El tiempo de ejecucion fue: 0.259150028229
```

Resultados digits

Tenemos 1275 patrones de entrenamiento, de los cuales 5%=64, 10%=128, 25%=319, 50%=638. A partir de estos datos del número de neuronas, obtendremos los valores del error cuadrático y precisión con media/desviación_típica.

Error cuadrático medio (Media/DT)

Num rbf	64		128		319		638	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	0.28/ 0.3	1.55/ 0.33	0/0	1.49/ 0.31	0/0	1.45/ 0.07	0/0	1.32/ 0.08

CCR % Precisión (Media/DT)

Num_rbf	64		128		319		638	
Eta	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
10e-5	98.8/ 0.16	91.8/ 1.32	100/0	94.2/ 0.94	100/0	93.9/ 0.3	100/0	94.4/ 0.3

En train podemos ver que se llega a alcanzar el 100% de clasificación, pero realmente nos interesa el valor de clasificación del test para poder generalizar. Se podría coger el valor para num_rbf=128 o 319, pero complicaría mucho el modelo y supondría tener muchos nodos en capa oculta. Por lo que escogeremos 64 para seguir investigando.

A continuación, para el modelo de num_rbf=64, modificaremos el parámetro eta para estudiar las modificaciones en los resultados.

Con regularización L1

	Error cuadrático medio				Precisión CCR %			
	Train		Test		Train		Test	
Eta	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**
0	0.294	0.023	1.57	0.136	98.8	0.08	91.66	0.8
0.1	1.33	0.16	1.66	0.24	93.8	0.64	90.8	0.41
0.01	0.73	0.08	1.02	0.2	96.7	0.38	93.5	0.7
0.001	0.3	0.09	1.46	0.25	98.6	0.34	91.29	0.72
0.0001	0.19	0.1	1.489	0.192	99.07	0.33	91.79	1.03
10e-5	0.23	0.04	1.39	0.27	98.73	0.16	92.1	1.33
10e-10	0.24	0.07	1.49	0.2	98.93	0.27	92.54	0.97

Para la regularización L1 podemos ver que con eta 0.01 se obtiene el mejor resultado en test. Esto se debe a que L1 considera pesos iguales a 0, y con esta poda realiza un buen resultado con tan solo 0.1.

Con regularización L2

	Error cuadrático medio				Precisión CCR %			
	Train		Test		Train		Test	
Eta	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**	**Media**	**DT**
0	0.22	0.05	1.93	0.63	98.89	0.15	90.53	1.05
0.1	2.01	0.05	1.43	0.02	90.09	0.56	89.40	0.53
0.01	1.38	0.14	1.21	0.24	93.66	0.5	91.35	0.83
0.001	0.95	0.1	1.1	0.16	95.95	0.34	92.98	0.31
0.0001	0.51	0.09	1.03	0.2	97.6	0.34	94.11	0.72
10e-5	0.52	0.1	1.42	0.22	97.6	0.38	92.41	0.6
10e-10	0.26	0.03	1.65	0.26	98.85	0.12	91.29	0.36

En este caso, es necesario considerar un eta 0.0001 para el mejor resultado porque L2 no considera valores iguales a 0. Por esto el valor C=1/eta tiene que ser mayor que para L1.

Por tanto nos quedamos con el valor eta=0.0001 y regularización L2. Con estos parámetros obtenemos un valor de CCR test del 94% y en entrenamiento alcanzamos casi 98%. Para el valor eta = 0.0001 vamos a comparar el número de coeficientes en capa oculta tanto para L1 como para L2.

```
[ -1.03153174e+01   1.91186313e+01   5.59384751e+01   5.13137731e+01
  -9.04362318e+01   8.51805703e+01   1.09048340e+02  -1.55833135e+02
  -9.59590554e+01   7.39873699e+00   1.52161227e+02   3.23184485e+01
  -8.53033827e+01  -3.49952742e+00  -5.69390074e+01  -2.66860582e+01
   1.75937768e+02   1.09819003e+02   4.88291760e+01  -7.37429337e+01
  -4.98745414e+00  -1.38721308e+01  -5.67435449e+01  -9.22611671e-01
   1.98643082e+01   0.00000000e+00  -7.13439563e+01  -5.02454469e+01
   2.76624614e+01  -9.11756782e+01  -7.73350156e+01  -1.82235522e+01
   8.24750057e+00  -3.06525633e+02   5.28521731e+00  -9.21767720e+00
```

Figura. Algunos pesos de los nodos de salida.

Tenemos 65 nodos, 64 rbf y uno de sesgo, para 10 clases de salida. Y podemos ver que algunos pesos son nulos para L1. Al contrario de lo que ocurre con L2.

En cuanto al tiempo computacional, hemos notado una gran mejoría en este algoritmo para la base de datos Digits. La ejecución en Python tarda unos escasos segundos, mientras que en CPP con la práctica 2 se extendía a varios minutos, resultaba bastante laborioso.

```
INFORME FINAL
-------------------------
Error de entrenamiento (Media +- DT): 0.548980 +- 0.141274
Error de test (Media +- DT): 1.253292 +- 0.173617
CCR de entrenamiento (Media +- DT): 97.47% +- 0.504802%
CCR de test (Media +- DT): 93.04% +- 0.415878%
El tiempo de ejecucion fue: 17.0083508492
MacBook-Air-de-Ruben-2:practica3 rubenmz$
```

Figura. Resultados obtenidos en la práctica3 con el script en Python

Y por último, vamos a mostrar la matriz de confusión resultante para los parámetros eta=0.0001 y L2. Y analizaremos los patrones que no estén bien clasificados. Podemos ver que esta matriz es bastante similar a la obtenida en la práctica 2 con la softmax de clasificación.

```
[[33  0  0  0  0  0  0  0  0  0]
 [ 0 29  1  1  1  0  0  0  0  1]
 [ 0  0 31  0  1  0  0  0  0  0]
 [ 0  0  0 29  0  1  0  0  0  2]
 [ 0  1  0  0 29  0  0  2  0  0]
 [ 0  0  1  0  0 30  0  0  0  0]
 [ 0  0  0  0  0  0 32  0  0  0]
 [ 0  2  0  0  1  0  0 27  1  1]
 [ 0  0  0  0  0  0  0  0 30  1]
 [ 0  0  1  1  0  0  0  1  0 28]]
```

Por ejemplo, hay dos 7 que están clasificados como 1, como este  o viceversa  . Esto ocurre normalmente, ya que la forma es similar entre ambos. Además, también un 9 que está clasificado como un 3, como  . También hay un 4 que está clasificado como un 7,  . Y así podemos continuar comparando dos a dos los dígitos ya que se asemejan en la forma.

5. REFERENCIAS BIBLIOGRÁFICAS

- **Documentación Scikit-Learn**
- **Funcion LogisticRegression**
- **Funcion KMeans**
- **Tutorial sobre Scikit-Learn**
- **Guión de la práctica**
- **Presentación de la práctica**