Redes de Neuronas

Práctica 2
Problema de Clasificación
Curso 2020-2021

Ramiro Leal Meseguer
100346124
100346124@alumnos.uc3m.es

Claudia Fernández Estebaranz
100366696
100366696@alumnos.uc3m.es

Índice

1. PARTE I

1.1. Introducción

En esta parte trabajaremos la clasificación de los tipos de cielo con el **perceptrón multicapa**. El objetivo es clasificar una serie de imágenes para hacer una estimación de si tenemos un cielo despejado, con un tipo de nube o varios tipos de nubes. El conjunto de datos de entrada, en este caso, está formado por 12 atributos que se corresponden con estadísticos extraídos de la imagen tomada, los cuales se obtienen a partir de los canales de color o por una transformación de dichos canales. Por otra parte, tendremos un atributo salida que nos indicará la clase a la que pertenece el cielo *(CieloDespejado, Nube, Multinube)*. Contaremos con 717 instancias formadas con estos 13 atributos para entrenar y testear el correcto funcionamiento del perceptrón.

1.2. Preparación de los datos

Antes a llevar a cabo el aprendizaje de las red, hay que realizar el procesado de los datos, para dicha tarea hemos decidido utilizar Python como lenguaje de programación.

Como nos marca el enunciado de la práctica, el procesado de datos tiene tres fases distintas:

- 1) Normalización de todos los valores en el rango [0,1] teniendo en cuenta los máximos y mínimos de cada atributo.
- 2) Preparación de los datos para Validación cruzada.
- 3) Finalmente todos los conjuntos deben aleatorizarse.

El fichero con el nombre *"Preparación_Datos.py"* es el programa encargado de realizar esta preparación de datos. Tras su ejecución, dados unos datos iniciales de entrada, generará, en la carpeta *"/DatosProcesados"*, 4 carpetas que serán los 4 folds que utilizaremos como datos de entrada para la red. Cada una de estas carpetas estará compuesta por su fichero de entrenamiento y test correspondiente. Además, también se generará el fichero *"datosNubes_MaximoMinimo.txt"* en el se guardarán los valores máximos y mínimos de cada una de las columnas del fichero de datos de entrada. Estos valores se van a utilizar posteriormente para la desnormalización de los datos una vez que se haya entrenado a la red.

1.2.1. Normalización

La fase de normalización consiste en cambiar los valores de los datos de entrada a una escala común sin distorsionar las diferencias entre los valores de diferentes rangos. Este proceso solo es necesario cuando las entradas están en distintas escalas.

Al ejecutar nuestro fichero *"Preparación_Datos.py"* estamos llevando la normalización de los datos de entrada en un rango de valores [0,1] y, además, generamos un fichero almacenando los máximos y mínimos de cada columna para después poder hacer el proceso inverso y desnormalizarlos.

1.2.2. Validación cruzada

En esta práctica aplicaremos la técnica de validación cruzada estratificada de 4 folds. Es por esto que debemos dividir el conjunto de datos en cuatro partes iguales (P1, P2, P3, P4), manteniendo siempre una proporción de instancias de las 3 clases *(CieloDespejado, Nube, Multinube)*. Una vez tengamos divididos los datos en 4 conjuntos procederemos a crear 4 parejas de ficheros:

- Fold 1: Entrenamiento: (P2 + P3 + P4), Test: P1
- Fold 2: Entrenamiento: (P1 + P3 + P4), Test: P2
- Fold 3: Entrenamiento: (P2 + P1 + P4), Test: P3

- Fold 4: Entrenamiento: (P2 + P3 + P1), Test: P4

Dado que el número de instancias de cada clase es:

- Clase cieloDespejado: 48
- Clase multinube: 156
- Clase nube: 513

Se dividieron las instancias de la siguiente manera para que la proporción de instancias de estas tres clases en las distintas partes sea igualitaria:

- cieloDespejado: 12 instancias a cada parte.
- multinube: 29 instancias a cada parte.
- nube: Dado que no se podía dividir en 4 partes iguales, habrá una de ellas, en concreto la P1, que contará con 129 instancias de nube y el resto de partes (P2, P3, P4) contarán con 128.

1.2.3. Aleatorización

Con la aleatorización se busca evitar cualquier posible sesgo durante el entrenamiento de la red.

1.3. Clasificación y pruebas

Para el estudio de clasificación con perceptrón multicapa utilizaremos el *simulador snns* bajo el lenguaje de programación R. Se nos facilitará un script inicial que entrenará al perceptrón, mostrará la gráfica de la evolución de los errores y calculará el porcentaje de aciertos. En nuestro caso, hemos añadido que nos muestre también el error cuadrático medio producido. En esta sección mostraremos un estudio de las diferentes arquitecturas que hemos probado, donde modificaremos la topología, la razón de aprendizaje, el número de ciclos y la seed. Todas las arquitecturas se probarán con los 4 folds, y a partir de esos 4 resultados realizaremos una media de ellas, obteniendo así un resultado medio de la arquitectura en sí.

Todas las pruebas realizadas las podemos encontrar en un .txt denominado ***PM - Pruebas***. En este fichero, se pueden observar todos los pasos que hemos seguido hasta conseguir el modelo final. En la memoria, solo incluiremos las más importantes para evitar extendernos.

Las primeras pruebas que hicimos fueron, a través de un modelo muy básico, tratar de encontrar la seed que mejor se ajustara a los resultados. Decidimos ajustar este parámetro el primero pues de esta manera comenzaremos desde el mejor punto de partida que nos permita encontrar el mínimo global/local. Una seed alejada de este mínimo puede producir que este no se encuentre y por ello, que el entrenamiento no sea correcto, obteniendo errores mayores. Tras múltiples pruebas, obtuvimos que aquella donde ***seed = 2*** daba los mejores resultados, es por esto, que todas las siguientes pruebas siempre tendrán esa seed.

Tras el ajuste de la *seed* procedimos al **análisis de la topología**, donde estudiaremos modelos de una, dos y tres capas con distinto número de neuronas en cada capa. Para estas pruebas siempre hemos utilizado una razón de aprendizaje = 0.01, unos ciclos máximos = 2000 y una seed = 2. Después de realizar distintas arquitecturas, hemos seleccionado los mejores modelos obtenido en cada caso (comentar que los resultados mostrados serán la media de los 4 folds):

Mejores modelos por topología				
Topología	**TrainMSE**	**accuraciesTrain**	**TestMSE**	**accuraciesTest**
c(7)	0,30200	0,78615	0,46877	0,60948
c(15)	0,30004	0,79219	0,47359	0,60392
c(20)	0,30387	0,78429	0,46845	**0,61507**
c(15,30)	0,29701	0,78847	0,46910	0,60114
c(20,30)	0,29712	0,78940	0,45751	**0,62066**
c(10, 15,20)	0,34214	0,39928	0,43002	0,30132
c(15, 20 , 30)	0,34152	0,40114	0,44644	0,29155

Podemos observar claramente que la topología con 3 capas no obtiene buenos resultados, es por esto, que ninguna de estas arquitecturas pasará a la siguiente fase. Entre las arquitecturas de una y dos capas ya empezamos a observar algunos resultados interesantes. Todos los modelos tienen resultados muy parecidos, pero hay dos de ellos que destacan un poco más por tener un mayor valor de porcentaje de acierto en el test. Estos modelos, serán **c(20) y C(20,30)**.
Si en este momento observamos la matriz de confusión obtenida del Test de las dos arquitecturas elegidas:

	1	**2**	**3**
1	6	0	6
2	0	4	35
3	0	30	99

(a) MCTest c(20)

	1	**2**	**3**
1	6	0	6
2	0	6	33
3	0	32	97

(b) MCTest c(20,30)

Cuadro 1: Matrices de confusión

Los resultados de ambas matrices son bastante similares. Podemos observar que el perceptrón donde peores resultados tiene es a la hora de predecir un cielo multiNube, la mayoría de las predicciones las confunde con nube. Este resultado es muy coherente pues son predicciones muy similares y pueden llevar a confusión.

A partir de estas topologías procederemos a realizar un **análisis de las razones de aprendizaje.** Probaremos distintas razones para estas dos topologías y seleccionaremos las mejores pruebas de cada una de estas. Comentar, que al igual que en las pruebas de topología aplicaremos una seed = 2 y un número máximo de ciclos de 2000. Dicho esto, en la siguiente tabla procedemos a mostrar las topologías junto con las razones de aprendizaje que mejores resultados obtuvieron:

Mejores modelos por topología					
Topología	**RA**	**TrainMSE**	**accuraciesTrain**	**TestMSE**	**accuraciesTest**
c(20)	**0,05**	0,23492	0,84334	**0,60126**	**0,57878**
c(20)	0,1	0,19608	0,87681	0,66260	0,57190
c(20)	0,2	0,20333	0,86565	0,67639	0,60668
c(20)	0,3	0,16130	0,89772	0,67981	0,61658
c(20)	0,4	0,15207	0,91307	0,67073	0,61935
c(20)	**0,7**	0,15041	0,91446	**0,662365**	**0,63047**
c(20,30)	0,05	0,21879	0,85263	0,59004	0,624937
c(20,30)	**0,2**	0,06054	0,96652	**0,61141**	**0,65973**
c(20,30)	**0,9**	0,08730	0,94699	**0,61136**	**0,65142**

Como podemos observar en la tabla, en el caso de una única capa encontramos los mejores modelos con una razón de aprendizaje de 0,05 y 0,7. En este caso, consideramos la arquitectura de 0,7 mejor que la de 0,05, pues nos proporciona el valor más alto de aciertos en el test. Podríamos haber elegido

la arquitectura con razón 0,05 dado que nos ofrece el mejor error de test, sin embargo, su porcentaje de aciertos es uno de los más bajos dentro de los mejores seleccionados. En el caso de dos capas, hemos encontrado menos modelos óptimos, sin embargo, hay dos que son similares y además, destacan sobre los demás, aquellos con razón de aprendizaje 0,2 y 0,9. Si tuviéramos que elegir entre uno de estos dos, nos decantaríamos por el de 0,2 dado que si sumamos el porcentaje de aciertos en el entrenamiento y el error, estos resultados también son mejores en esta arquitectura. Por tanto, finalmente seleccionaremos **c(20) con RA = 0,7 y c(20,30) con RA= 0,2**. Para elegir, así a primera vista, entre estos dos modelos, por los resultados de la tabla anterior, elegiríamos el modelo con dos capas. Sin embargo, para verificar que esta elección es correcta, procederemos también a mirar la matrices de confusión de ambos:

	1	**2**	**3**
1	6	0	6
2	0	14	25
3	0	55	74

(a) MCTest c(20) R.A = 0,7

	1	**2**	**3**
1	6	0	6
2	0	13	26
3	0	35	94

(b) MCTest c(20,30) R.A = 0.2

Cuadro 2: Matrices de confusión

Observando las matrices podemos ver que la predicción de cielo es igual para ambos. Por otra parte, ambos modelos tienen errores a la hora de diferenciar entre nube y multinube, sin embargo, la **arquitectura c(20,30) con razón de aprendizaje 0,2** comete menos fallos, es por esto, que nuestra primera previsión de este como modelo final, era correcta.

Por tanto, podemos decir que nuestro **modelo final** esta formado por los **siguientes hiperparámetros**:

- Topología: C(20, 30)
- Seed = 2
- Razón de aprendizaje = 0,2
- Máximo número de ciclos = 2000

Desglosamos los resultados que nos ofrece esta arquitectura.

Modelo Final				
Fold	**TrainMSE**	**accuraciesTrain**	**TestMSE**	**accuraciesTest**
1	0,05956	0,97020	0,67815	0,62777
2	0,07893	0,95353	0,49385	0,72625
3	0,03889	0,97955	0,45699	0,73184
4	0,06476	0,96282	0,81664	0,55307
Media	0,06054	0,96652	0,61141	0,65973

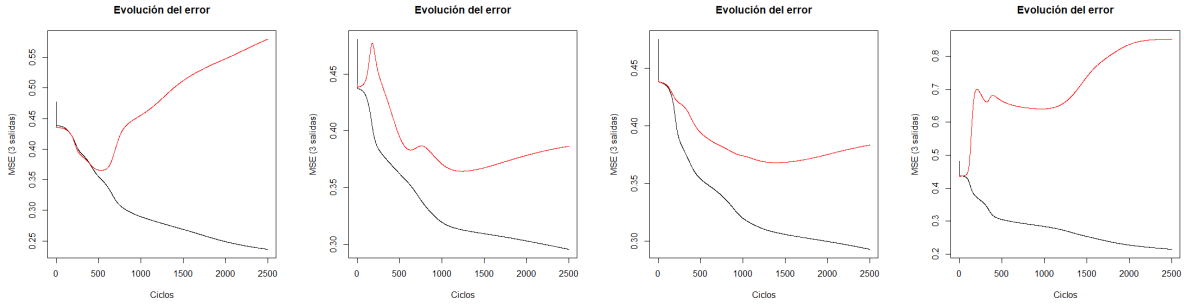
	1	**2**	**3**
1	33	0	3
2	0	84	33
3	6	35	344

(a) MCTrain

	1	**2**	**3**
1	6	0	6
2	0	13	26
3	0	35	94

(b) MCTest

Cuadro 3: Matrices de confusión



(a) Fold 1 **(b)** Fold 2 **(c)** Fold 3 **(d)** Fold 2

Figura 1: Evolución errores

1.4. Conclusiones

Como conclusión podemos decir que obtenemos buenos resultados, tenemos un 96 % de aciertos en el entrenamiento, aunque luego se reduce a un 66 % en el test. Es cierto, que este valor se puede considerar algo bajo, pero al menos el acierto es mayor al 50 %. Como hemos comentado antes, además, si miramos las matrices de confusión final, vemos que donde más casos de error tenemos es entre el tipo Nube y MultiNube, debido a su similitud, es muy complicado diferenciarlas lo que puede conllevar a una confusión de ambos casos de la red. Para mejorar estos valores, quizás lo más conveniente sería aumentar la base de datos, aunque por lo que demuestra la red el aprendizaje casi es perfecto por lo que la mejora de esta es complicada.

Al margen de los resultados, que hemos comentado a lo largo de la práctica, queremos comentar aspectos más personales sobre posibles errores cometidos. Es cierto, que esta práctica era muy similar a la anterior, más o menos se seguía el mismo esquema. Hemos tratado de llevar un análisis exhaustivo de las pruebas dado que en este caso no era necesario el desarrollo de código, por lo que lo consideramos primordial. Debido a los buenos resultados que tuvimos en la anterior práctica, hemos seguido un poco la misma metodología de pruebas tratando de considerar todos los casos que sean relevantes. No hemos incluido todas las pruebas en la memoria pues hay algunas que los resultados no eran buenos, pero todas estas se pueden encontrar en el fichero mencionado en la sección de pruebas. Esta práctica ha llevado mucho esfuerzo en este caso solo para las pruebas, aunque queremos comentar que inicialmente, para ahorrarnos este trabajo tratamos de modificar el código proporcionado para tener uno que nos devolviera más resultados en una ejecución. Comentar, que debido al ajuste de tiempo (por el resto de asignaturas) no pudimos profundizar en este aspecto y acabamos realizando las pruebas con el código inicial.

Referencias

[1] UC3M. Enunciado práctica 2 parte i. https://aulaglobal.uc3m.es/pluginfile.php/4108999/mod_resource/content/1/Practica2-parte1_2020-21.pdf, 2020.