Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Desarrollo de aplicaciones avanzadas de ciencias computacionales (Gpo 301)

Actividad 2.1.: Entrenamiento de una RNA

Autor:

Daniel Esteban Maldonado Espitia - A01657967

26 de mayo del 2023

1. **A partir de la base de datos de diabetes, implementar un proceso de balanceo en la BD, de tal forma que queden igual número de instancias por clase (se recomienda 1000 instancias, pero puede manejarse otro número).**

Para realizar el proceso de balanceo en la BD, decidí emplear la técnica "Synthetic Minority Oversampling Technique" o SMOTE que ya viene disponible dentro de Weka como uno de los filtros que se pueden aplicar a los datos para hacer el sobremuestreo de una clase en específico.

Para llegar a una base de datos con 1000 instancias por clase, se requirió hacer un aumento del 100% de las instancias negativas, ya que se contaba con 500, y se hizo un aumento del 273.15% de las instancias positivas, ya que se contaba con 268.

Esta base de datos balanceada se guardó en el archivo *act2.1_diabetes_2000.arff*.

2. **A partir de la base de datos balanceada, particionar la misma de tal forma que se divida la base de datos en un conjunto training y un conjunto test global. Se puede manejar una partición 80 - 20, o bien una partición 70 - 30, por ejemplo. Generar dos archivos en los cuales se almacenen las particiones generadas. Se debe cuidar que las particiones se construyan con un proceso estratificado - estocástico, cuidando el balanceo por clases en cada partición.**

Ahora bien, antes de hacer la partición de los datos, se aplicó el filtro de Randomize para evitar que las particiones quedaran sesgadas con datos de una sola clase. Para lograr esto, se aplicó el dicho filtro en Weka 3 veces, usando como semilla los números primos 41, 67 y 97. La BD resultante se guardó en el archivo *act2.1_diabetes_2000_rand.arff*.

Luego, para generar las particiones del conjunto training y test global, se decidió hacer una repartición del 80% para el training y el 20% para el test. Decidí emplear el filtro de StratifiedRemoveFolds que tiene Weka, ya que no solo permite hacer las particiones de forma personalizada, sino que también las construye con un proceso estratificado - estocástico, lo que permite que haya el mismo número de instancias de clase positivas y negativas en cada pliegue.

Para generar los archivos correspondientes, se indicaron los siguientes valores en los parámetros del filtro de Weka: el fold que se extraería sería el fold 1, se mantuvo el invertSelection en False para extraer el conjunto Test y True para extraer el conjunto Training, y se indicó que se harían 5 pliegues en numFolds ya que así se obtendrían folds con el 20% de los datos cada uno.

La partición con el conjunto de Training se almacenó en el archivo *act2.1_diabetes_2000_rand_training80.arff*, mientras que el conjunto test se guardó en el archivo *act2.1_diabetes_2000_rand_test20.arff*.

3. **Sobre el conjunto training global, aplicar un proceso de validación cruzada K = 3. Cuidar que la representatividad de los patrones se mantenga lo más posible en cada partición (generar un conjunto training y un conjunto validation por cada pliegue, guardando los archivos generados).**

Para aplicar el proceso de validación cruzada sobre el conjunto training global, primero se aplicó un filtro aleatorio dentro de dicho archivo con semillas 41, 67 y 97, y se almacenó el resultado en el archivo *act2.1_diabetes_2000_rand_training80_rand.arff*.

Luego, se empleó el filtro de StratifiedRemoveFolds para obtener los siguientes pliegues:

- Para el conjunto test del pliegue 1:
    - Parámetros de StratifiedRemoveFolds: fold = 1, invertSelection = False, numFolds = 3
    - Archivo en donde se guardó: act2.1_diabetes_2000_rand_training80_rand_fold1_test.arff.
- Para el conjunto training del pliegue 1:
    - Parámetros de StratifiedRemoveFolds: fold = 1, invertSelection = True, numFolds = 3
    - Archivo en donde se guardó: act2.1_diabetes_2000_rand_training80_rand_fold1_training.arff.
- Para el conjunto test del pliegue 2:
    - Parámetros de StratifiedRemoveFolds: fold = 2, invertSelection = False, numFolds = 3

    - Archivo en donde se guardó: act2.1_diabetes_2000_rand_training80_rand_fold1_test.arff.
- Para el conjunto training del pliegue 2:
    - Parámetros de StratifiedRemoveFolds: fold = 2, invertSelection = True, numFolds = 3
    - Archivo en donde se guardó: act2.1_diabetes_2000_rand_training80_rand_fold2_training.arff.
- Para el conjunto test del pliegue 3:
    - Parámetros de StratifiedRemoveFolds: fold = 3, invertSelection = False, numFolds = 3
    - Archivo en donde se guardó: act2.1_diabetes_2000_rand_training80_rand_fold3_test.arff.
- Para el conjunto training del pliegue 3:
    - Parámetros de StratifiedRemoveFolds: fold = 3, invertSelection = True, numFolds = 3
    - Archivo en donde se guardó: act2.1_diabetes_2000_rand_training80_rand_fold3_training.arff.

4. **Conducir un proceso experimental para construir un modelo de aprendizaje basado en una RNA sobre cada pliegue de la validación cruzada. Aplicar la técnica de la rejilla para la exploración de los hiper – parámetros de ajuste de la RNA. Se deberá cuidar qué para cada experimento, el resultado que se obtenga no esté sobre – ajustado (overfitting), o bien quede en sub – ajuste (underfitting). Generar las gráficas correspondientes para mostrar en qué momento el entrenamiento entra en overfitting.**

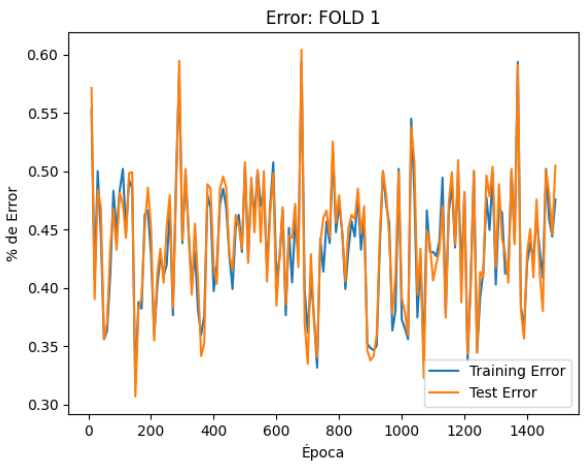
Ahora bien, para construir un modelo de aprendizaje basado en una RNA, decidí emplear código de Python en donde, haciendo uso del “Multi-layer Perceptron classifier” que implementa la librería de Scikit-learn, fuera obteniendo los valores de ajuste de la red en distintas épocas y esos los graficara para observar el momento en el que hay overfitting de los datos.

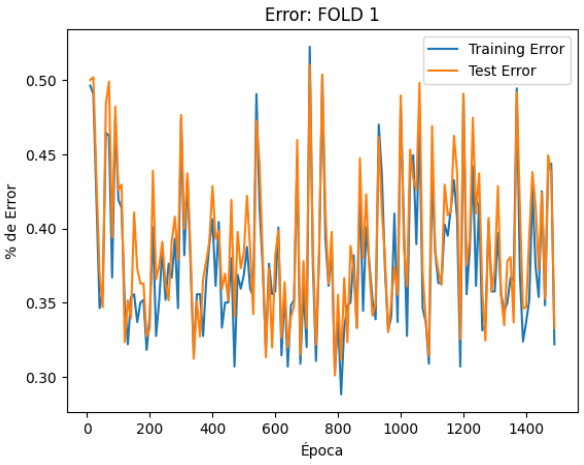
Para ello, primero realicé una exploración para determinar cuáles son los mejores hiper - parámetros de ajuste aplicando la técnica de la rejilla, tomando como referencia las siguientes recomendaciones realizadas por el profesor:

- Para el número de neuronas se debe comenzar con valores altos y explorar hacia abajo y hacia arriba; por lo que se probará en un rango de 7 a 12 neuronas.
- Para las capas ocultas, se debe iniciar en 1 e ir probando hasta 3 o 4; por lo que se probará en un rango de 1 a 3.
- Para el número de épocas se debe comenzar en pocas épocas, de 20 a 50, e ir subiendo hasta donde ya no haya sobreajuste (overfitting) o subajuste (underfitting); por lo que se probará en un rango de 20 a 300.
- Para el learning rate se debe comenzar en 0.3 e ir probando en el rango [0.1, 0.4].
- Para el Momentum se debe comenzar en 0.2 y luego moverlo en el rango [0.0, 0.3].

Una vez teniendo definidos los valores entre los que se estará aplicando el método de rejilla, empecé a probarlos con el MLPClassifier. Primero, realicé pruebas con el solver SGD ya que es el único que permite ajustar los parámetros de learning rate y momentum. Sin embargo, al hacer las pruebas con este solver, las gráficas generadas nunca tuvieron un comportamiento adecuado en donde el error tanto de training como de test fueran disminuyendo periódicamente y luego llegan a overfitting (el training sigue bajando mientras que el test se mantiene o sube).

Ejemplos de las gráficas generadas son este solver se muestran a continuación:





Al investigar el motivo por el cual esto estaba ocurriendo, encontré que los solvers LBFGS es el que mejor se adapta a datasets pequeños, y al estar trabajando con 2000 instancias en total, este solver resultaba el más adecuado para usar. Por esta razón y después de hacer las mismas

pruebas con el solver ADAM, el cuál hubiera permitido ajustar el learning rate, se decidió utilizar el solver LBFGS, con el cuál los únicos hiper parámetros que se podrán ajustar son el número de neuronas, el número de capas ocultas y el número de épocas.

Por lo que, al aplicar la técnica de la rejilla sobre el Fold 1, se obtuvieron los siguientes resultados:

```
(12,12,12)
overfitting claro
época overfitting = 75
valores error bajos (época 70): training = 0.265, test = 0.282
DIFF = 0.017

(12,12)
overfitting claro
época overfitting = 100
valores error bajos (época 80): training = 0.285, test = 0.32
DIFF = 0.035

(12)
overfitting claro (mostrando hasta época 500)
época overfitting = 320
valores error bajos (época 300): training = 0.248, test = 0.249
DIFF = 0.001

(11,11,11)
overfitting claro (mostrando hasta época 500)
época overfitting = 250
valores error (época 250): training = 0.208, test = 0.238
DIFF = 0.03

(11,11) # Descartado: no baja a más de 0.35 el error
sin overfitting claro

(11)
overfitting claro (mostrando hasta época 500)
época overfitting = 400
valores error (época 360): training = 0.236, test = 0.237
DIFF = 0.001

(10,10,10)
overfitting claro (mostrando hasta época 500)
época overfitting = 170
valores error bajos (época 140): training = 0.255, test = 0.258
DIFF = 0.003

(10,10)
overfitting claro (mostrando hasta época 500)
época overfitting = 220
valores error bajos (época 210): training = 0.238, test = 0.247
DIFF = 0.009

(10)
overfitting claro (mostrando hasta época 600)
época overfitting = 410
```

```
valores error bajos (época 380): training = 0.193, test = 0.206
DIFF = 0.013

(9,9,9)
overfitting claro (mostrando hasta época 500)
época overfitting = 220
valores error bajos (época 187): training = 0.284, test = 0.295
DIFF = 0.011

(9,9)
UNDERFITTING (mostrando hasta época 500)
época underfitting = 210
valores error bajos (época 168): training = 0.273, test = 0.267
DIFF = 0.006

(9)
UNDERFITTING (mostrando hasta época 200)
época underfitting = 125
valores error bajos (época 124): training = 0.268, test = 0.268
DIFF = 0

(8,8,8) # Descartado: UNDERFITTING y error no baja de 0.25
UNDERFITTING (mostrando hasta época 200)

(8,8) # No genera un modelo adecuado ni graficable

(8) # No genera un modelo adecuado ni graficable

(7,7,7) # Descartado: error muy alto antes de overfitting
overfitting claro (mostrando hasta época 500)
época overfitting = 130
valores error bajos (época 35): training = 0.34, test = 0.37
DIFF = 0.03

(7,7) # Descartado: error muy alto antes de overfitting
overfitting claro (mostrando hasta época 200)
época overfitting = 120
valores error bajos (época 35): training = 0.31, test = 0.33
DIFF = 0.02

(7)
overfitting claro (mostrando hasta época 1000)
época overfitting = 780
valores error bajos (época 770): training = 0.217, test = 0.222
DIFF = 0.005
```

Explicación: Los números al principio de cada párrafo representan las capas ocultas y el número de neuronas que tiene cada una, mientras que el DIFF hasta es el valor de diferencia entre el valor de error de training y test.

Una vez teniendo estos resultados, se puede empezar a descartar los valores más bajos de diferencia y los errores más altos, para encontrar el modelo que mejor se adapte a nuestras necesidades. Al observar que hay varios modelos que nos dieron valores de error tanto en el

training como en el test menores a 0.25 podemos descartar todos los que tienen un valor mayor a este, lo cual nos deja con los siguientes elementos:

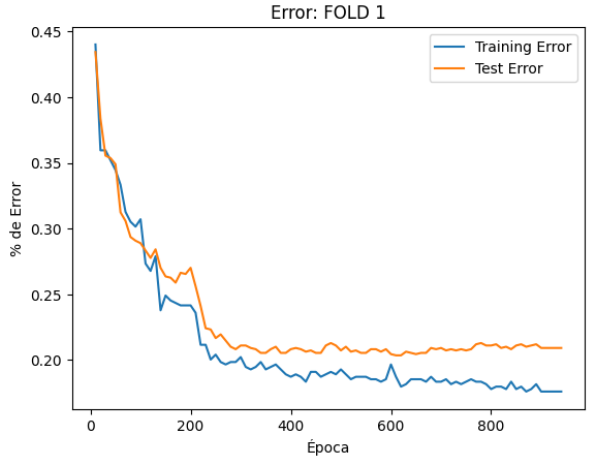
```
(11)
overfitting claro (mostrando hasta época 500)
época overfitting = 400
valores error (época 360): training = 0.236, test = 0.237
DIFF = 0.001

(10)
overfitting claro (mostrando hasta época 600)
época overfitting = 410
valores error bajos (época 380): training = 0.193, test = 0.206
DIFF = 0.013

(7)
overfitting claro (mostrando hasta época 1000)
época overfitting = 780
valores error bajos (época 770): training = 0.217, test = 0.222
DIFF = 0.005
```

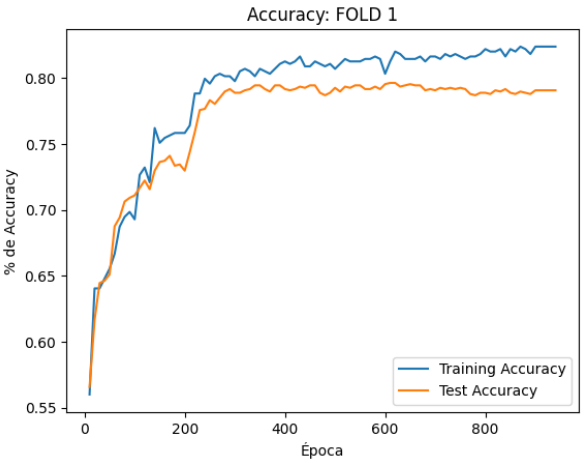
Después de también descartar los valores que tenían una diferencia alta, nos quedamos con los 3 de arriba.

Ahora bien, a pesar de que el primer valor sea el que mejor diferencia tiene, sus errores no son tan bajos como en los otros dos casos, y si medimos la diferencia entre el error del training y test del modelo con 10 neuronas con respecto a las demás obtenemos valores que nos demuestran cómo se ajusta de mejor manera este modelo a los datos, a pesar de que su diferencia sea mayor. Por esta razón, se decidió usar un modelo de 8 neuronas, 1 capa oculta y 380 épocas.

Estas gráficas muestran el porcentaje accuracy y error obtenido con este modelo, y muestran como más o menos en la época 410 es donde empieza el overfitting:

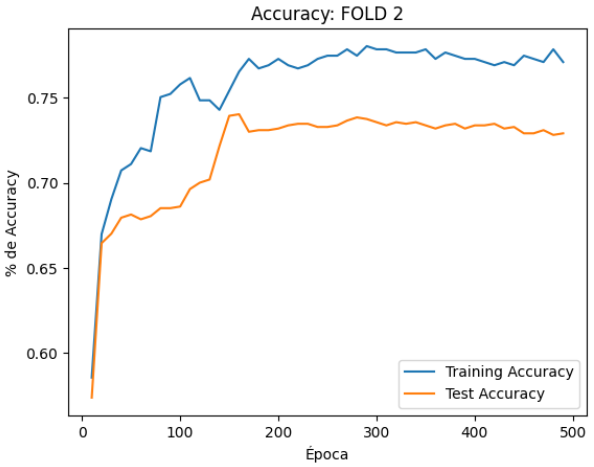
FOLD 1:



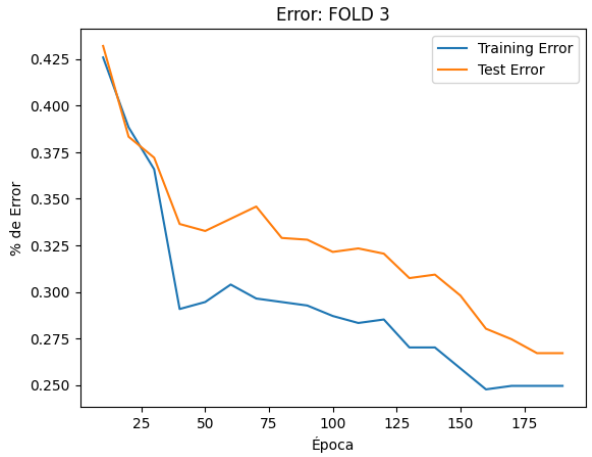


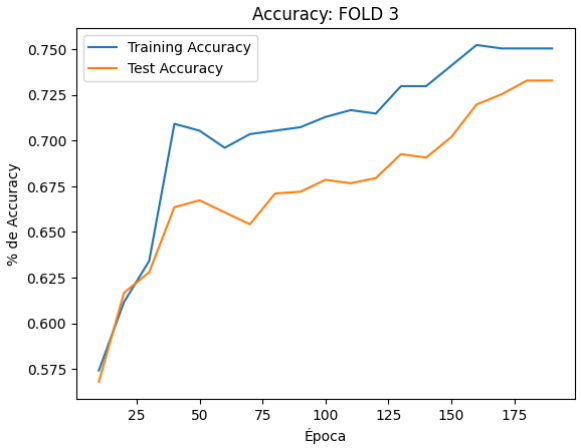
Los resultados con dichos hiper parámetros en los demás folds fueron los siguientes:

FOLD 2: Entra en overfitting en la época 180.



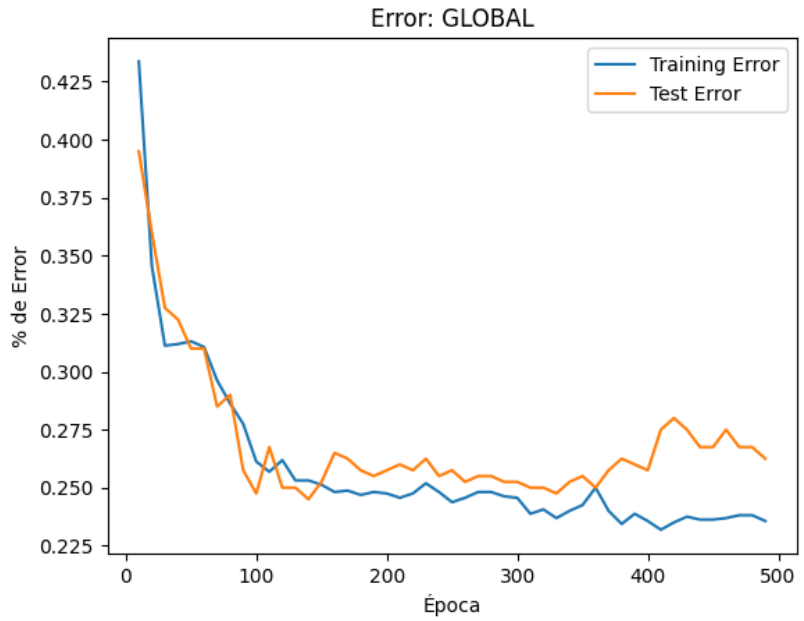
FOLD 3: Entra en overfitting en la época 180.

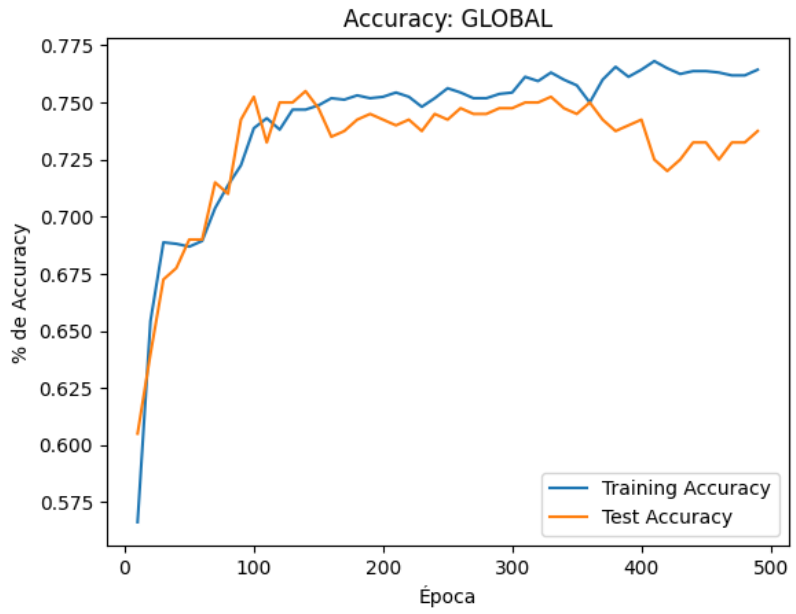




5. **Realizar un experimento final en el cual se entrene una RNA (con búsqueda de hyper - parámetros) sobre el conjunto training y test global, cuidando que los resultados no queden en underfitting ni en overfitting. Hacer un análisis comparativo con respecto a los resultados obtenidos en los pliegues, de tal forma que se discuta que tan robustos son los resultados obtenidos.**

Por último, al correr el modelo con los mismos hiper parámetros determinados previamente, se obtuvieron los siguientes resultados:





Con ello, se obtuvo como valor de accuracy óptimo (más alto antes de overfitting) un 74.5% tanto en training como en test.

Con este resultado final, se puede observar que los hiper parámetros encontrados con el método de la rejilla son bastante adecuados, ya que nos permiten llegar a un valor de accuracy relativamente alto, pero sobretodo porque nos permite tener ese mismo valor entre el conjunto training y test en una época en específico; es decir, en la época 360 obtenemos tanto en el training como en el test un valor de 74.5% de accuracy.

Ahora bien, comparando estos resultados con los obtenidos en los pliegues de la validación cruzada podemos observar que los valores de accuracy solo llegan a ser más altos en el fold 1, dónde el training y test llegan a valores mayores a 79%, pero en todos los demás casos obtenemos valores de accuracy en el rango de 72% a 75%.

Esto nos lleva a la conclusión de que la Red Neuronal Artificial fue entrenada con hiper parámetros que son mejores para un fold en específico, pero que para los demás se mantienen congruentes los resultados. Por esto, puedo concluir que el modelo generado tendría la capacidad de predecir de buena manera el si un paciente tiene diabetes o no a partir de los valores predictores.

El código implementado en Python para generar los modelos y gráficas se encuentra completo dentro del zip con los pliegues.