Práctica 3: Redes neuronales de funciones de base radial

Convocatoria de febrero (curso académico 2018/2019)

Asignatura: Introducción a los modelos computacionales
4º Grado Ingeniería Informática (Universidad de Córdoba)

8 de noviembre de 2018

Resumen

Esta práctica sirve para familiarizar al alumno con el concepto de red neuronal de funciones de base radial (RBF). De esta forma, desarrollaremos un código que entrene una red de este tipo, utilizando Python y la librería de aprendizaje automático `scikit-learn`[1]. De este modo, la práctica servirá para familiarizarse con librerías externas, que tan a menudo son necesarias en entornos de aprendizaje automático. El alumno deberá programar el algoritmo y comprobar el efecto de distintos parámetros sobre un conjunto de bases de datos reales. La entrega se hará utilizando la tarea en Moodle habilitada al efecto. Se deberá subir en un único fichero comprimido todos los entregables indicados en este guión. El día tope para la entrega es el **28 de noviembre de 2018**. En caso de que dos alumnos entreguen prácticas copiadas, no se puntuarán ninguna de las dos.

1. Introducción

El trabajo que se va a realizar en la práctica consiste en implementar una red neuronal de tipo RBF realizando un entrenamiento en tres etapas:

1. Aplicación de un algoritmo de *clustering* que servirá para establecer los centros de las funciones RBF (pesos de capa de entrada a capa oculta).
2. Ajuste de los radios de las RBF, mediante una heurística simple (media de las distancias hacia el resto de centros).
3. Aprendizaje de los pesos de capa oculta a capa de salida.
   - Para problemas de regresión, utilización de la pseudo-inversa de Moore Penrose.
   - Para problemas de clasificación, utilización de un modelo lineal de regresión logística.

El alumno deberá desarrollar un *script* de Python capaz de realizar el entrenamiento de una red RBF con las características anteriormente mencionadas. Este *script* se utilizará para entrenar modelos que predigan de la forma más correcta posible un conjunto de bases de datos disponible en Moodle y se realizará un análisis de los resultados obtenidos. **Este análisis influirá en gran medida en la calificación de la práctica**.

En el enunciado de esta práctica, se proporcionan valores orientativos para todos los parámetros del algoritmo. Sin embargo, se valorará positivamente si el alumno encuentra otros valores para estos parámetros que le ayuden a mejorar los resultados obtenidos.

La sección 2 describe una serie de pautas generales a la hora de implementar el algoritmo de entrenamiento de redes neuronales de tipo RBF. La sección 3 explica los experimentos a realizar

[1] `http://scikit-learn.org/`

una vez implementado el algoritmo. Finalmente, la sección 4 especifica los ficheros a entregar para esta práctica.

2. Implementación del algoritmo de entrenamiento de redes RBF

2.1. Arquitectura de los modelos a considerar

Los modelos de redes neuronales RBF que vamos a considerar tienen la siguiente arquitectura:

- Una capa de entrada con tantas neuronas como variables tenga la base de datos considerada.
- Una capa oculta con un número de neuronas a especificar por el usuario del *script* a desarrollar. Es importante recalcar que, en las dos prácticas anteriores, el número de capas ocultas era variable. En esta práctica **siempre tendremos una sola capa oculta**. Todas las neuronas de la capa oculta serán de tipo RBF (en contraposición a las neuronas de tipo sigmoide, utilizadas en prácticas anteriores).
- Una capa de salida con tantas neuronas como variables de salida tenga la base de datos considerada:
  - Si la base de datos es de **regresión**, todas las neuronas de la capa de salida serán de tipo lineal (igual que las neuronas de tipo sigmoide, pero sin aplicar la transformación $\frac{1}{1+e^{-x}}$).
  - Si la base de datos es de **clasificación**, todas las neuronas de la capa de salida serán de tipo *softmax*. No hay que implementar la transformación *softmax*, ya que esta ya está implementada por el algoritmo de regresión logística que utilizaremos para ajustar los pesos de la capa de salida.

2.2. Ajuste de los pesos

Se deben de seguir las indicaciones aportadas en las diapositivas de clase para que el entrenamiento se realice de la siguiente forma

1. Aplicación de un algoritmo de *clustering* que servirá para establecer los centros de las funciones RBF (pesos de capa de entrada a capa oculta). Para problemas de clasificación, la inicialización de los centroides se realizará seleccionando aleatoriamente, y de forma estratificada, n_1 patrones[2]. Para problemas de regresión, seleccionaremos aleatoriamente n_1 patrones. Después de inicializar los centroides, para realizar el *clustering*, utilizaremos la clase `sklearn.cluster.KMeans`, con una sola inicialización de los centroides (`n_init`) y un máximo de 500 iteraciones (`max_iter`).

2. Ajuste de los radios de las RBF, mediante una heurística simple (la mitad de la media de las distancias hacia el resto de centros). Es decir, el radio de la neurona j será[3]:

$$\sigma_j = \frac{1}{2 \cdot (n_1 - 1)} \sum_{i \neq j} \|c_j - c_i\| = \frac{1}{2 \cdot (n_1 - 1)} \sum_{i \neq j} \sqrt{\sum_{d=1}^{n} (c_{jd} - c_{id})^2}. \quad (1)$$

[2]Para esta labor, puedes consultar la clase `sklearn.model_selection.StratifiedShuffleSplit`, que realiza una o varias particiones de una base de datos de forma "estratificada", es decir, manteniendo la proporción de patrones de cada clase en la base de datos original
`http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html`

[3]Considera el uso conjunto de las funciones `pdist` y `squareform` de `scipy` para obtener la matriz de distancias

3. Aprendizaje de los pesos de capa oculta a capa de salida.

   - Para problemas de regresión, utilización de la pseudo-inversa de Moore Penrose. Es decir:

$$\boldsymbol{\beta}^{\mathrm{T}}_{((n_1+1)\times k)} = \quad (\mathbf{R}^{+})_{((n_1+1)\times N)}\,\mathbf{Y}_{(N\times k)} = \tag{2}$$

$$= \quad \left(\mathbf{R}^{\mathrm{T}}_{((n_1+1)\times N)} \times \mathbf{R}_{(N\times(n_1+1))}\right)^{-1} \mathbf{R}^{\mathrm{T}}_{((n_1+1)\times N)}\mathbf{Y}_{(N\times k)} \tag{3}$$

     dónde $\mathbf{R}$ es la matriz que contiene las salidas de las neuronas RBF, $\boldsymbol{\beta}$ es una matriz conteniendo un vector de parámetros por cada salida a predecir e $\mathbf{Y}$ es una matriz con todas las salidas deseadas. Para realizar estas operaciones, utilizaremos las funciones matriciales de `numpy`, que es una de las dependencias de `scikit-learn`.

   - Para problemas de clasificación, utilización de un modelo lineal de regresión logística. Haremos uso de la clase `sklearn.linear_model.LogisticRegression`, aportando un valor para el parámetro C que aplica regularización. Es necesario indicar que en esta librería lo que especificamos es el valor de coste C (importancia del error de aproximación frente al error de regularización), de forma que $\eta = \frac{1}{C}$. Utilizaremos la expresión de regularización de tipo L2[4] y el algoritmo de optimización `liblinear`.

3. Experimentos a realizar

Probaremos distintas configuraciones de la red neuronal y ejecutaremos cada configuración con cinco semillas (100, 200, 300, 400 y 500). A partir de los resultados obtenidos, se obtendrá la media y la desviación típica del error. Para problemas de regresión mostraremos el error de tipo MSE. Para problemas de clasificación, mostraremos el error de tipo MSE[5]y, además, el *script* deberá mostrar el porcentaje de patrones bien clasificados o CCR.

Para valorar cómo funciona el algoritmo implementado en esta práctica, emplearemos tres bases de datos de regresión:

- *Función seno*: esta base de datos está compuesta por 120 patrones de *train* y 41 patrones de *test*. Ha sido obtenido añadiendo cierto ruido aleatorio a la función seno (ver Figura 1).
- *Base de datos quake*: esta base de datos está compuesta por 1633 patrones de *train* y 546 patrones de *test*. Se corresponde con una base de datos en la que el objetivo es averiguar la fuerza de un terremoto (medida en escala sismológica de Richter). Como variables de entrada, utilizamos la profundidad focal, la latitud en la que se produce y la longitud [6].
- *Base de datos parkinsons*: esta base de datos está compuesta por 4406 patrones de *train* y 1469 patrones de *test*. Contiene, como entradas o variables independientes, una serie de datos clínicos de pacientes con la enfermedad de Parkinson y datos de medidas biométricas de la voz, y, como salidas o variables dependientes, el valor motor y total del UPDRS (de las siglas en inglés *Unified Parkinson's Disease Rating Scale*)[7].

Y dos bases de datos de clasificación:

- *Base de datos vote*: *vote* contiene 326 patrones de entrenamiento y 109 patrones de test. La base de datos incluye los votos para cada uno de los para cada uno de los candidatos para el Congreso de los EEUU, identificados por la CQA. Todas las variables de entrada son categóricas[8].

[4] `https://msdn.microsoft.com/en-us/magazine/dn904675.aspx`

[5] En clasificación, el MSE debe obtenerse como se hizo en la práctica 2, es decir, convirtiendo las etiquetas de clase a valores binarios y comparándolos con las probabilidades predichas, que pueden obtenerse con el método `predict_proba`

[6] Para más información, consultar `https://sci2s.ugr.es/keel/dataset.php?cod=75`

[7] Para más información, consultar `http://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring`

[8] Para más información, consultar `hhttps://archive.ics.uci.edu/ml/datasets/congressional+voting+records`

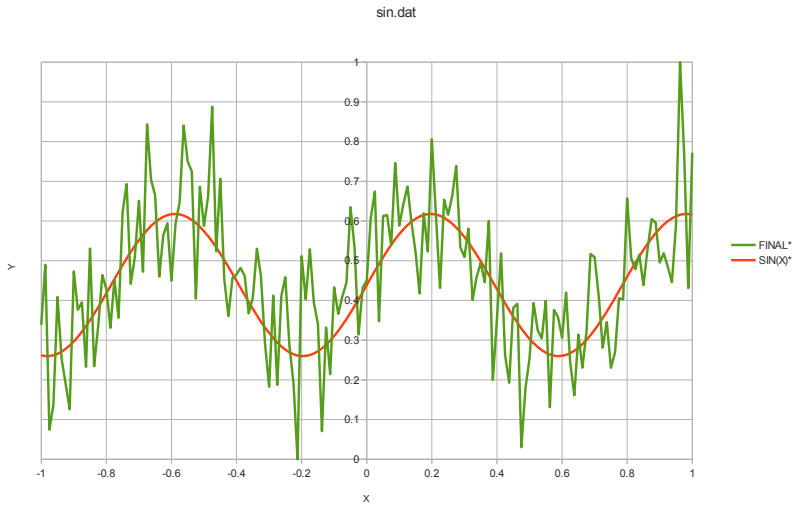

Figura 1: Representación de los datos incluidos para el problema de estimación de la función seno.

- *Base de datos noMNIST*: esta base de datos, originariamente, está compuesta por $200{,}000$ patrones de entrenamiento y $10{,}000$ patrones de *test*, y un total de 10 clases. No obstante, para la práctica que nos ocupa, se ha reducido considerablemente el tamaño de la base de datos para realizar las pruebas en menor tiempo. Por lo tanto la base de datos que se utilizará está compuesta por 900 patrones de entrenamiento y 300 patrones de *test*. Está formada por un conjunto de letras (de la a a la f) escritas con difererentes tipografías o simbologías. Están ajustadas a una rejilla cuadrada de 28×28 píxeles. Las imágenes están en escala de grises en el intervalo $[-1{,}0; +1{,}0]$.[9]. Cada uno de los píxeles forman parte de las variables de entrada (con un total de $28 \times 28 = 784$ variables de entrada) y las clases se corresponden con la letra escrita (a, b, c, d, e y f, con un total de 6 clases). La figura 2 representa un subconjunto de los 180 patrones del conjunto de entrenamiento, mientras que la figura 3 representa un subconjunto de 180 letras del conjunto de *test*. Además, todas las letras, ordenadas dentro de cada conjunto, está colgadas en la plataforma Moodle en los ficheros `train_img_nomnist.tar.gz` y `test_img_nomnist.tar.gz`.

Figura 2: Subconjunto de letras del conjunto de entrenamiento.

Se deberá extraer la media y desviación típica de dos medidas (en regresión) o cuatro medidas (en clasificación):

- Regresión: media y desviación típica del MSE de entrenamiento y de *test*.

[9]Para más información, consultar `http://yaroslavvb.blogspot.com.es/2011/09/notmnist-dataset.html`

Figura 3: subconjunto de letras del conjunto de *test*.

- Clasificación: media y desviación típica del CCR de entrenamiento y de *test* y media y desviación típica del MSE de entrenamiento y de *test*. El MSE que se pide para el caso de clasificación es el que se obtiene cuando comparamos las salidas deseadas (0 para las clases incorrectas y 1 para la clase correcta) con las probabilidades predichas por el modelo (lo que se conoce como *Brier score*[10]).

Se deben probar, al menos, las siguientes configuraciones:

- *Arquitectura de la red*:
  - Para todas las bases de datos, considerar un número de neuronas en capa oculta (n_1) igual al $5\,\%$, $10\,\%$, $25\,\%$ y $50\,\%$ del número de patrones de la base de datos. En esta fase, para problemas de clasificación, utilizar regularización L1 y un valor para el parámetro $\eta = 10^{-5}$.
- Para los problemas de clasificación, una vez decidida la mejor arquitectura, probar los siguientes valores para η: $\eta = 1$, $\eta = 0{,}1$, $\eta = 0{,}01$, $\eta = 0{,}001$, ..., $\eta = 10^{-10}$, junto con los dos tipos de regularización (L2 y L1). ¿Qué sucede?. Calcula la diferencia en número de coeficientes en vote y noMNIST cuando modificas el tipo de regularización (L2 Vs L1)[11].
- Para problemas de regresión y de clasificación, comparar los resultados obtenidos con la inicialización propuesta para el algoritmo `sklearn.cluster.KMeans` (usando la mejor arquitectura y la mejor configuración para la regresión logística) con respecto a la inicialización `'k-means++'`.
- Finalmente, en alguno de los problemas de clasificación, probar a lanzar el *script* considerando el problema como si fuera un problema de regresión (es decir, incluyendo un `False` en el parámetro `clasificacion` y calculando el CCR redondeando las predicciones hasta el entero más cercano). ¿Qué sucede en este caso?.

Como valor orientativo, se muestra a continuación el error de entrenamiento y de generalización obtenido por una regresión lineal utilizando Weka en las tres bases de datos:

- *Función seno*: $MSE_{\text{train}} = 0{,}02968729$; $MSE_{\text{test}} = 0{,}03636649$.
- *Base de datos Quake*: $MSE_{\text{train}} = 0{,}03020644$; $MSE_{\text{test}} = 0{,}02732409$.
- *Base de datos Parkinsons*: $MSE_{\text{train}} = 0{,}043390$; $MSE_{\text{test}} = 0{,}046354$.

y el CCR de entrenamiento y de generalización obtenido por una regresión logística lineal utilizando Weka en las dos bases de datos de clasificación:

- *Base de datos vote*: $CCR_{\text{entrenamiento}} = 96{,}0123\,\%$; $CCR_{\text{test}} = 92{,}6606\,\%$.

[10] `https://en.wikipedia.org/wiki/Brier_score`

[11] Los coeficientes están en el atributo `coef_` del objeto que realiza la regresión logística. Considerar que si el valor absoluto de un coeficiente es menor que 10^{-5} entonces el coeficiente es nulo

- *Base de datos noMNIST*: $CCR_{\text{entrenamiento}} = 80{,}4444\,\%; CCR_{\text{test}} = 82{,}6667\,\%$.

El alumno debería ser capaz de superar estos valores con algunas de las configuraciones y semillas.

3.1. Formato de los ficheros

Los ficheros que contienen las bases de datos tendrán formato CSV, de forma que los valores vendrán separados por comas. En este caso, no tendremos cabeceras. Para realizar la lectura de los ficheros, utilizaremos la función `read_csv` de la librería `pandas`.

4. Entregables

Los ficheros a entregar serán los siguientes:

- Memoria de la práctica en un fichero `pdf` que describa el *script* generado, incluya las tablas de resultados y analice estos resultados.
- *Script* de Python correspondiente a la práctica.

4.1. Memoria de la práctica

La memoria de la práctica deberá incluir, al menos, el siguiente contenido:

- Portada con el número de práctica, título de la práctica, asignatura, titulación, escuela, universidad, curso académico, nombre, DNI y correo electrónico del alumno.
- Índice del contenido de la memoria con numeración de las páginas.
- Descripción de los pasos a realizar para llevar a cabo el entrenamiento de las redes RBF (**máximo 1 carilla**).
- Experimentos y análisis de resultados:
  - Breve descripción de las bases de datos utilizadas.
  - Breve descripción de los valores de los parámetros considerados.
  - Resultados obtenidos, según el formato especificado en la sección anterior.
  - Análisis de resultados. El análisis deberá estar orientado a justificar los resultados obtenidos, en lugar de realizar un análisis meramente descriptivo de las tablas. Tener en cuenta que esta parte es decisiva en la nota de la práctica. Se valorará la inclusión de los siguientes elementos de comparación:
    - Matriz de confusión en test del mejor modelo de red neuronal obtenido para la base de datos *noMNIST*. Analizar los errores cometidos, incluyendo las imágenes de aquellos caracteres en los que el modelo de red se equivoca, para comprobar si son confusos. Comparación de esta matriz con la matriz obtenida para el perceptrón multicapa en la práctica anterior.
    - Tiempo computacional necesario para entrenar la base de datos `noMNIST` y comparativa con el tiempo necesario para la práctica anterior.
- Referencias bibliográficas u otro tipo de material distinto del proporcionado en la asignatura que se haya consultado para realizar la práctica (en caso de haberlo hecho).

Aunque lo importante es el contenido, se valorará también la presentación, incluyendo formato, estilo y estructuración del documento. La presencia de demasiadas faltas ortográficas puede disminuir la nota obtenida.

4.2. Código fuente

Junto con la memoria, se deberá incluir el *script* de Python preparado para funcionar en las máquinas de la UCO (en concreto, probar por `ssh` en `ts.uco.es`). El *script* a desarrollar deberá recibir los siguientes argumentos por línea de comandos[12]:

- Argumento `-t`, `--train_file`: Indica el nombre del fichero que contiene los datos de entrenamiento a utilizar. Sin este argumento, el programa no puede funcionar.
- Argumento `-T`, `--test_file`: Indica el nombre del fichero que contiene los datos de $test$ a utilizar. Si no se especifica este argumento, utilizar los datos de entrenamiento como $test$.
- Argumento `-c`, `--classification`: Booleano que indica si el problema es de clasificación. Si no se especifica, supondremos que el problema es de regresión.
- Argumento `-r`, `--ratio_rbf`: Indica la razón (en tanto por uno) de neuronas RBF con respecto al total de patrones en entrenamiento. Si no se especifica, utilizar $0{,}1$ capa oculta.
- Argumento `-l`, `--l2`: Booleano que indica si utilizaremos regularización de L2 en lugar de la regularización L1. Si no se especifica, supondremos que regularización L1.
- Argumento `-e`, `--eta`: Indica el valor del parámetro eta (η). Por defecto, utilizar $\eta = 1e-2$.
- Argumento `-o`, `--outputs`: Indica el número de columnas de salida que tiene el conjunto de datos y que siempre están al final. Por defecto, utilizar $o = 1$.
- (Kaggle) Argumento `-p`, `--pred`: Booleano que indica si utilizaremos el modo de predicción.
- (Kaggle) Argumento `-m`, `--model_file`: Indica el directorio en el que se guardarán los modelos entrenados (en el modo de entrenamiento, sin el *flag* p) o el fichero que contiene el modelo que se utilizará (en el modo de predicción, con el *flag* p).
- Argumento `--help`: Mostrar la ayuda del programa (utilizar la que genera automáticamente la librería `click`).

Un ejemplo de ejecución de dicho *script* puede verse en la siguiente salida [13]:

```
i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py --help
Usage: rbf.py [OPTIONS]

  Modelo de aprendizaje supervisado mediante red neuronal de tipo RBF.
  Ejecución de 5 semillas.

Options:
  -t, --train_file TEXT  Fichero con los datos de entrenamiento.
  -T, --test_file TEXT   Fichero con los datos de test.  [required]
  -c, --classification   Activar para problemas de clasificación.  [default:
  False]
  -r, --ratio_rbf FLOAT  Ratio (en tanto por uno) de neuronas RBF con respecto
  al total de patrones.  [default: 0.1]
  -l, --l2               Activar para utilizar la regularización de L2 en
  lugar de la regularización L1 (regresión logística).
  [default: False]
  -e, --eta FLOAT        Valor del parámetro de regularización para la
  regresión logística.  [default: 0.01]
  -o, --outputs INTEGER  Número de variables que se tomarán como salidas
```

[12] Para procesar la secuencia de entrada, se utilizará la librería `click`

[13] Para que el código funcione en las máquinas de la UCO, tendrás que instalar los paquetes `click` y la última versión de `scikit-learn`, utilizando los comandos:
`pip install scikit-learn --user --upgrade`
`pip install click --user --upgrade`

```
  (todas al final de la matriz).  [default: 1]
  -m, --model_file TEXT  Fichero en el que se guardará o desde el que se
  cargará el modelo (si existe el flag p).  [default: ]
  -p, --pred             Activar el modo de predicción.  [default: False]
  --help                 Show this message and exit.

i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -t ./csv/train_vote.csv -T ./csv/
    test_vote.csv -c -r 0.1 -e 0.001 --l2
-----------
Semilla: 100
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.010718
MSE de test: 0.042227
CCR de entrenamiento: 98.47%
CCR de test: 95.41%
-----------
Semilla: 200
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.011173
MSE de test: 0.041298
CCR de entrenamiento: 98.47%
CCR de test: 93.58%
-----------
Semilla: 300
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.007499
MSE de test: 0.040842
CCR de entrenamiento: 99.08%
CCR de test: 96.33%
-----------
Semilla: 400
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.016981
MSE de test: 0.030527
CCR de entrenamiento: 97.85%
CCR de test: 96.33%
-----------
Semilla: 500
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.013229
MSE de test: 0.035569
CCR de entrenamiento: 98.47%
CCR de test: 96.33%
**********
Resumen de resultados
**********
MSE de entrenamiento: 0.011920 +- 0.003126
MSE de test: 0.038093 +- 0.004439
CCR de entrenamiento: 98.47% +- 0.39%
CCR de test: 95.60% +- 1.07%

i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -t ./csv/train_parkinsons.csv -T ./
    csv/test_parkinsons.csv -r 0.5 -o 2
-----------
Semilla: 100
-----------
Número de RBFs utilizadas: 2203
MSE de entrenamiento: 0.005099
MSE de test: 0.050147
-----------
Semilla: 200
-----------
```

```
Número de RBFs utilizadas: 2203
MSE de entrenamiento: 0.005051
MSE de test: 0.054609
-----------
Semilla: 300
-----------
Número de RBFs utilizadas: 2203
MSE de entrenamiento: 0.005072
MSE de test: 0.050170
-----------
Semilla: 400
-----------
Número de RBFs utilizadas: 2203
MSE de entrenamiento: 0.005252
MSE de test: 0.052640
-----------
Semilla: 500
-----------
Número de RBFs utilizadas: 2203
MSE de entrenamiento: 0.005317
MSE de test: 0.046462
**********
Resumen de resultados
**********
MSE de entrenamiento: 0.005158 +- 0.000106
MSE de test: 0.050806 +- 0.002740

i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -t ./csv/train_parkinsons.csv -T ./
    csv/test_parkinsons.csv -r 0.1 -o 2
-----------
Semilla: 100
-----------
Número de RBFs utilizadas: 440
MSE de entrenamiento: 0.016788
MSE de test: 0.020662
-----------
Semilla: 200
-----------
Número de RBFs utilizadas: 440
MSE de entrenamiento: 0.016857
MSE de test: 0.020919
-----------
Semilla: 300
-----------
Número de RBFs utilizadas: 440
MSE de entrenamiento: 0.016803
MSE de test: 0.021173
-----------
Semilla: 400
-----------
Número de RBFs utilizadas: 440
MSE de entrenamiento: 0.016992
MSE de test: 0.021033
-----------
Semilla: 500
-----------
Número de RBFs utilizadas: 440
MSE de entrenamiento: 0.016873
MSE de test: 0.021128
**********
Resumen de resultados
**********
MSE de entrenamiento: 0.016863 +- 0.000072
MSE de test: 0.020983 +- 0.000182

i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -t ./csv/train_sin.csv -T ./csv/
    test_sin.csv -r 0.1 -o 1
```

```
-----------
Semilla: 100
-----------
Número de RBFs utilizadas: 12
MSE de entrenamiento: 0.012661
MSE de test: 0.034499
-----------
Semilla: 200
-----------
Número de RBFs utilizadas: 12
MSE de entrenamiento: 0.012521
MSE de test: 0.030468
-----------
Semilla: 300
-----------
Número de RBFs utilizadas: 12
MSE de entrenamiento: 0.012538
MSE de test: 0.034354
-----------
Semilla: 400
-----------
Número de RBFs utilizadas: 12
MSE de entrenamiento: 0.012536
MSE de test: 0.035553
-----------
Semilla: 500
-----------
Número de RBFs utilizadas: 12
MSE de entrenamiento: 0.012720
MSE de test: 0.024141
**********************
Resumen de resultados
**********************
MSE de entrenamiento: 0.012595 +- 0.000080
MSE de test: 0.031803 +- 0.004203

# Aquí estamos lanzando clasificación como si fuese regresión
i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -t ./csv/train_vote.csv -T ./csv/
    test_vote.csv -r 0.1
-----------
Semilla: 100
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.032015
MSE de test: 0.044310
-----------
Semilla: 200
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.031910
MSE de test: 0.038348
-----------
Semilla: 300
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.032739
MSE de test: 0.041902
-----------
Semilla: 400
-----------
Número de RBFs utilizadas: 32
MSE de entrenamiento: 0.036787
MSE de test: 0.043404
-----------
Semilla: 500
-----------
Número de RBFs utilizadas: 32
```

```
MSE de entrenamiento: 0.030798
MSE de test: 0.049926
**********
Resumen de resultados
**********
MSE de entrenamiento: 0.032850 +- 0.002064
MSE de test: 0.043578 +- 0.003769
CCR de entrenamiento: 96.75 % +- 0.31 %
CCR de test: 94.50 % +- 0.82 %
```

4.3. [OPCIONAL] Guardar el modelo en un fichero.

Durante la ejecución del entrenamiento, el *script* permite guardar el modelo entrenado en un fichero `pickle`[14]. Esto permitirá utilizar el modelo entrenado para predecir las salidas del conjunto de datos de **Kaggle**.

Para guardar el modelo, será necesario utilizar el parámetro `-m`. A continuación se muestra un ejemplo de ejecución:

```
i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -t train.csv -T test.csv -l -c -r 0.1
    -m modelo
-----------
Semilla: 100
-----------
Número de RBFs utilizadas: 400
MSE de entrenamiento: 0.053684
MSE de test: 0.059213
CCR de entrenamiento: 77.75 %
CCR de test: 75.13 %
-----------
Semilla: 200
-----------
Número de RBFs utilizadas: 400
MSE de entrenamiento: 0.052957
MSE de test: 0.058708
CCR de entrenamiento: 77.75 %
CCR de test: 75.30 %
-----------
Semilla: 300
-----------
Número de RBFs utilizadas: 400
MSE de entrenamiento: 0.052909
MSE de test: 0.058892
CCR de entrenamiento: 78.05 %
CCR de test: 75.23 %
-----------
Semilla: 400
-----------
Número de RBFs utilizadas: 400
MSE de entrenamiento: 0.052931
MSE de test: 0.058543
CCR de entrenamiento: 77.78 %
CCR de test: 75.10 %
-----------
Semilla: 500
-----------
Número de RBFs utilizadas: 400
MSE de entrenamiento: 0.052372
MSE de test: 0.058348
CCR de entrenamiento: 78.30 %
CCR de test: 75.17 %
**********
Resumen de resultados
**********
```

[14] https://docs.python.org/3/library/pickle.html

```
MSE de entrenamiento: 0.052970 +- 0.000418
MSE de test: 0.058741 +- 0.000297
CCR de entrenamiento: 77.93% +- 0.22%
CCR de test: 75.19% +- 0.07%
```

Cuando finalice la ejecución, tendremos una carpeta llamada "modelo" que contendrá 5 ficheros `pickle`. Cada uno de ellos se corresponde con el modelo generado para cada semilla. A la hora de obtener predicciones, se deberá escoger uno de estos 5 ficheros.

```
i02gupep@NEWTS:~/imc/workspace/practica3$ ls modelo/
 1.pickle  2.pickle  3.pickle  4.pickle  5.pickle
```

4.4. [OPCIONAL] Obtener predicciones para Kaggle.

Una vez que se ha guardado el modelo en un fichero, es posible obtener las predicciones de las salidas para el conjunto de Kaggle. Para ello, se debe hacer uso de los parámetros `-m` y `-p`. A continuación se muestra un ejemplo:

```
i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -T kaggle.csv -p -m modelo/2.pickle
 Id,Category
 0,4
 1,0
 2,2
 3,0
 4,0

 ...

 2995,4
 2996,3
 2997,2
 2998,5
 2999,1
```

Para mayor facilidad, se puede redirigir la salida a un fichero `csv`:

```
i02gupep@NEWTS:~/imc/workspace/practica3$ ./rbf.py -T kaggle.csv -p -m modelo/2.pickle >
     predicciones.csv
```

Este fichero está listo para subirlo a Kaggle.