Práctica 3: Redes neuronales de funciones de base radial

Convocatoria de enero (curso académico 2020/2021)

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

3 de noviembre de 2020

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 **23 de noviembre de 2020**. 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 el método `sklearn.model_selection.train_test_split`, 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
`https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.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:

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

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

     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 (1, 2, 3, 4 y 5). 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 divorce*: *divorce* contiene 127 patrones de entrenamiento y 43 patrones de test. La base de datos contiene la respuesta a una serie de preguntas de un conjunto de encuestas en las que se pretende predecir si se va a producir un divorcio en la pareja. Las respuestas a las preguntas de la encuesta se proporcionan en escala de Likert con valores de 0 a 4. Todas

[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`

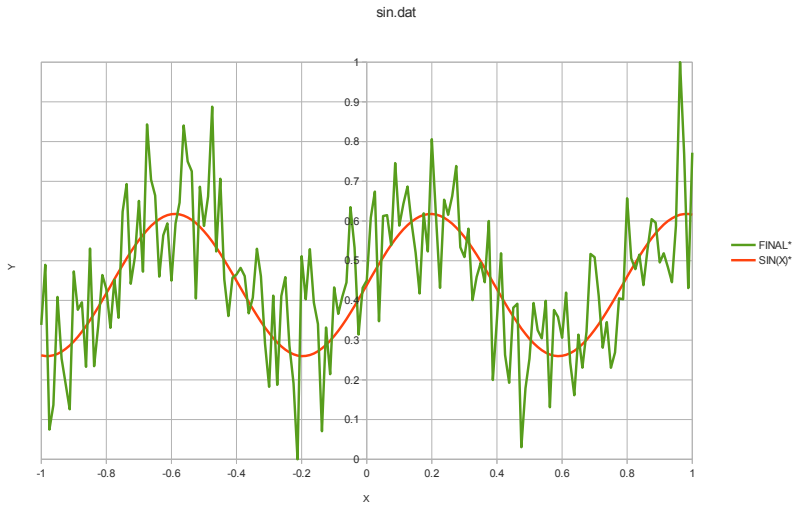

Figura 1: Representación de los datos incluidos para el problema de estimación de la función seno.

las variables de entrada se consideran numéricas. A continuación se muestran dos ejemplos de preguntas asociadas a la encuesta:

- *23. I know my spouse's favorite food.*
- *24. I can tell you what kind of stress my spouse is facing in her/his life.*

La base de datos tiene un total de 54 preguntas (por lo tanto, 54 variables de entrada) y dos categorías (0 no hay divorcio, 1 hay divorcio)[8].

- *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`.

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*.

[8]Para más información, consultar `https://archive.ics.uci.edu/ml/datasets/Divorce+Predictors+data+set`

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

Figura 2: Subconjunto de letras del conjunto de entrenamiento.

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 $5\,\%$, $15\,\%$, $25\,\%$ 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 divorce 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?.

[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

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 divorce*: $CCR_{\text{entrenamiento}} = 90{,}5512\,\%$; $CCR_{\text{test}} = 90{,}6977\,\%$.
- *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]:

[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
```

```
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py --help
Usage: rbf.py [OPTIONS]

  5 executions of RBFNN training

  RBF neural network based on hybrid supervised/unsupervised training. We
  run 5 executions with different seeds.

Options:
  -t, --train_file TEXT  Name of the file with training data.
  -T, --test_file TEXT   Name of the file with test data.  [required]
  -c, --classification   The problem considered is a classification problem.
                         [default: False]
  -r, --ratio_rbf FLOAT  Ratio of RBF neurons (as a fraction of 1) with
                         respect to the total number of patterns.  [default:
                         0.1]
  -l, --l2               Use L2 regularization instead of L1 (logistic
                         regression).  [default: False]
  -e, --eta FLOAT        Value of the regularization parameter for logistic
                         regression.  [default: 0.01]
  -o, --outputs INTEGER  Number of columns that will be used as target
                         variables (all at the end).  [default: 1]
  -p, --pred             Use the prediction mode.  [default: False]
  -m, --model TEXT       Directory to save the model (or name of the
                         file to load the model, if the prediction mode is
                         active).
  --help                 Show this message and exit.



i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./csv/train_divorce.csv -T ./csv/
    test_divorce.csv -c -r 0.15 -e 0.001 --l2
-----------
Seed: 1
-----------
Number of RBFs used: 19
Training MSE: 0.000112
Test MSE: 0.001935
Training CCR: 100.00 %
Test CCR: 100.00 %
-----------
Seed: 2
-----------
Number of RBFs used: 19
Training MSE: 0.000099
Test MSE: 0.018344
Training CCR: 100.00 %
Test CCR: 97.67 %
-----------
Seed: 3
-----------
Number of RBFs used: 19
Training MSE: 0.000135
Test MSE: 0.007341
Training CCR: 100.00 %
Test CCR: 97.67 %
-----------
Seed: 4
-----------
Number of RBFs used: 19
Training MSE: 0.000130
Test MSE: 0.017479
Training CCR: 100.00 %
Test CCR: 97.67 %
-----------
Seed: 5
-----------
```

```
Number of RBFs used: 19
Training MSE: 0.000352
Test MSE: 0.011236
Training CCR: 100.00 %
Test CCR: 97.67 %
******************
Summary of results
******************
Training MSE: 0.000166 +- 0.000094
Test MSE: 0.011267 +- 0.006183
Training CCR: 100.00 % +- 0.00 %
Test CCR: 98.14 % +- 0.93 %

# En los siguientes ejemplos, los CCRs salen 0 porque es un problema de regresión
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./csv/train_parkinsons.csv -T ./csv/
    test_parkinsons.csv -r 0.5 -o 2
-----------
Seed: 1
-----------
Number of RBFs used: 2203
Training MSE: 0.005435
Test MSE: 0.061848
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 2
-----------
Number of RBFs used: 2203
Training MSE: 0.005209
Test MSE: 0.055629
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 3
-----------
Number of RBFs used: 2203
Training MSE: 0.005230
Test MSE: 0.051494
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 4
-----------
Number of RBFs used: 2203
Training MSE: 0.005305
Test MSE: 0.060224
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 5
-----------
Number of RBFs used: 2203
Training MSE: 0.005250
Test MSE: 0.051680
Training CCR: 0.00 %
Test CCR: 0.00 %
******************
Summary of results
******************
Training MSE: 0.005286 +- 0.000081
Test MSE: 0.056175 +- 0.004266
Training CCR: 0.00 % +- 0.00 %
Test CCR: 0.00 % +- 0.00 %

i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./csv/train_parkinsons.csv -T ./csv/
    test_parkinsons.csv -r 0.15 -o 2
-----------
```

```
Seed: 1
-----------
Number of RBFs used: 660
Training MSE: 0.013441
Test MSE: 0.019442
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 2
-----------
Number of RBFs used: 660
Training MSE: 0.014156
Test MSE: 0.019407
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 3
-----------
Number of RBFs used: 660
Training MSE: 0.014024
Test MSE: 0.020129
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 4
-----------
Number of RBFs used: 660
Training MSE: 0.014096
Test MSE: 0.019187
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 5
-----------
Number of RBFs used: 660
Training MSE: 0.014192
Test MSE: 0.020314
Training CCR: 0.00 %
Test CCR: 0.00 %
******************
Summary of results
******************
Training MSE: 0.013982 +- 0.000276
Test MSE: 0.019696 +- 0.000442
Training CCR: 0.00 % +- 0.00 %
Test CCR: 0.00 % +- 0.00 %

i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./csv/train_sin.csv -T ./csv/test_sin.
    csv -r 0.15 -o 1
-----------
Seed: 1
-----------
Number of RBFs used: 18
Training MSE: 0.012100
Test MSE: 0.104196
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 2
-----------
Number of RBFs used: 18
Training MSE: 0.011401
Test MSE: 0.200121
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 3
```

```
-----------
Number of RBFs used: 18
Training MSE: 0.011954
Test MSE: 0.102267
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 4
-----------
Number of RBFs used: 18
Training MSE: 0.012082
Test MSE: 0.083309
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 5
-----------
Number of RBFs used: 18
Training MSE: 0.011961
Test MSE: 0.092522
Training CCR: 0.00 %
Test CCR: 0.00 %
******************
Summary of results
******************
Training MSE: 0.011899 +- 0.000257
Test MSE: 0.116483 +- 0.042481
Training CCR: 0.00 % +- 0.00 %
Test CCR: 0.00 % +- 0.00 %

# Aquí estamos lanzando clasificación como si fuese regresión
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./csv/train_divorce.csv -T ./csv/
    test_divorce.csv -r 0.15
-----------
Seed: 1
-----------
Number of RBFs used: 19
Training MSE: 0.016020
Test MSE: 0.020228
Training CCR: 97.64 %
Test CCR: 97.67 %
-----------
Seed: 2
-----------
Number of RBFs used: 19
Training MSE: 0.014577
Test MSE: 0.020006
Training CCR: 98.43 %
Test CCR: 97.67 %
-----------
Seed: 3
-----------
Number of RBFs used: 19
Training MSE: 0.014949
Test MSE: 0.018446
Training CCR: 98.43 %
Test CCR: 97.67 %
-----------
Seed: 4
-----------
Number of RBFs used: 19
Training MSE: 0.012619
Test MSE: 0.021317
Training CCR: 98.43 %
Test CCR: 97.67 %
-----------
Seed: 5
```

```
-----------
Number of RBFs used: 19
Training MSE: 0.016418
Test MSE: 0.021326
Training CCR: 97.64 %
Test CCR: 97.67 %
******************
Summary of results
******************
Training MSE: 0.014917 +- 0.001332
Test MSE: 0.020265 +- 0.001059
Training CCR: 98.11 % +- 0.39 %
Test CCR: 97.67 % +- 0.00 %
```

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/la3$ ./rbf.py -t train.csv -T test.csv -l -c -r 0.01 -m
    model
-----------
Seed: 1
-----------
Number of RBFs used: 118
Training MSE: 0.152570
Test MSE: 0.155294
Training CCR: 31.97 %
Test CCR: 28.87 %
-----------
Seed: 2
-----------
Number of RBFs used: 118
Training MSE: 0.152697
Test MSE: 0.155242
Training CCR: 31.70 %
Test CCR: 28.21 %
-----------
Seed: 3
-----------
Number of RBFs used: 118
Training MSE: 0.152596
Test MSE: 0.155267
Training CCR: 31.88 %
Test CCR: 28.58 %
-----------
Seed: 4
-----------
Number of RBFs used: 118
Training MSE: 0.152599
Test MSE: 0.155124
Training CCR: 31.87 %
Test CCR: 28.79 %
-----------
Seed: 5
-----------
Number of RBFs used: 118
Training MSE: 0.152681
Test MSE: 0.155183
```

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

```
Training CCR: 31.51 %
Test CCR: 28.78 %
******************
Summary of results
******************
Training MSE: 0.152629 +- 0.000051
Test MSE: 0.155222 +- 0.000061
Training CCR: 31.78 % +- 0.16 %
Test CCR: 28.65 % +- 0.24 %
```

Cuando finalice la ejecución, tendremos una carpeta llamada "model" 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/la3$ ls model/
 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/la3$ ./rbf.py -T kaggle.csv -p -m model/2.pickle
Id,Category
0,4
1,4
2,3
3,4
4,4
5,1
6,3
7,4
8,0


 ...

13859,0
13860,4
13861,2
13862,0
13863,3
13864,3
13865,0
13866,2
13867,3
13868,3
13869,0
13870,0
13871,1
13872,4
13873,4
13874,3
13875,4
```

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

```
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -T kaggle.csv -p -m modelo/2.pickle >
    submission.csv
```

Este fichero está listo para subirlo a Kaggle.