Lab assignment 3: Radial basis functions neural networks

Academic year 2023/2024

Subject: Introduction to computational models
4th course Computer Science Degree (University of Córdoba)

15th November 2023

Resumen

This lab assignment serves as familiarisation for the student with radial basis functions (RBF) neural networks. In this way, a RBF neural network will be developed, using Python and the `scikit-learn` library [1]. In this sense, the assignment will also serve as familiarisation with external libraries, widely used in machine learning (`numpy`, `pandas`...). In addition, we will introduce the problem of bias in machine learning models through `fairlearn`[2]. The student must implement the algorithm and analyse the effect of different parameters over a given set of real-world datasets. Delivery will be made using the task in Moodle authorized for this purpose. All deliverables must be uploaded in a single compressed file indicated in this document. The deadline for the submission is **12th December 2023**. In case two students submit copied assignments, neither of them will be scored.

1. Introduction

The work to be done in this lab assignment consists in implementing a RBF neural network with a training stage divided into three steps:

1. Application of a clustering algorithm which will be used to establish the centres of the RBF (input-to-hidden-layer's weights).
2. The RBF radium adjustment is done by means of a simple heuristic (distance average to the rest of the centres).
3. Hidden-to-output's weights learning:
    - For regression problems, using the Moore-Penrose's pseudo-inverse.
    - For classification problems, using a logistic regression linear model.

The student should develop a Python's script able to train a RBF neural network with the aforementioned characteristics. This programme will be used to train models able to classify as accurate as possible a set of databases available in Moodle. Also, an analysis about the obtained results will be included. For the COMPAS dataset that we studied in lab assignment 2, we will also perform an algorithmic bias analysis to contrast the behaviour of the models in different demographic groups. **This analysis will greatly influence the qualification of this assignment.**

In the statement of the assignment, indicative values are provided for all parameters. However, it will be positively evaluated if the student finds other values for these parameters able to achieve better results.

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

Section 2 describes a series of general guidelines when implementing the training algorithm for RBF neural networks. Section 3 explains the experiments to be carried out once the algorithm is implemented. Finally, section 4 specifies the files to be delivered for this assignment.

2. Implementation of the RBF neural network training algorithm

2.1. Model's architecture to be considered

The RBF neural network models should have the following architecture:

- An input layer with as many neurons as input variables the dataset has.
- A hidden layer with a number of neurons specified by the user. It is important to highlight that, in the two previous lab assignment, the number of hidden layer was variable. However, for this lab assignment, we are going to consider just one hidden layer. The type of all the neurons in the hidden layer will be RBF (in contrast to the sigmoidal neurons used in the previous lab assignments).
- An output layer with as many neurons as output variables the dataset has.
  - When considering regression datasets, all the output neurons will be linear (i.e. similar to the sigmoidal neurons without the application of the $\frac{1}{1+e^{-x}}$ transformation).
  - When considering classification datasets, all the output neurons will be softmax. The softmax function is already implemented by the logistic regression algorithm used for adjusting the weights of the output layer.

2.2. Weights adjustment

The instructions given in the class slides should be followed so that the training is carried out as follows:

1. Application of a clustering algorithm that will serve to establish the centres of the RBF (input-to-output layer weights). For classification problems, the centroid initialisation will be random and stratified, n_1 patterns[3]. For regression problems, n_1 will be randomly selected. After initialising the centroids, the `sklearn.cluster.KMeans` class will be used, with only one centroid initialisation (`n_init`) and a maximum of 500 iterations (`max_iter`).
2. To adjust the radium of the RBF, a simple heuristic will be applied (the half of the distance average to the rest of the centres). This is, the radium of the j-th neuron will be[4]:

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

3. Learning the weights from hidden-to-output layer.
   - For regression problem, it is done using the Moore-Penrose pseudo-inverse. This is:

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

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

[3]For this, the `sklearn.model_selection.train_test_split` method can be used. It performs one or more stratified dataset partitions, this is, keeping the ratio of patterns belonging to each class in the original dataset `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

[4]Consider using the functions `pdist` and `squareform` of `scipy` to obtain the distances matrix

where $\mathbf{R}$ is the matrix containing the outputs of the RBF neurons, $\boldsymbol{\beta}$ is a matrix containing a vector of parameters for each of the outputs to be predicted, and $\mathbf{Y}$ is a matrix with the target outputs. To perform these operations, we will use the matrix functions of `numpy`, which is a dependence of `scikit-learn`.

- For classification problems, it is done using a logistic regression linear model. Using the `sklearn.linear_model.LogisticRegression` class, providing a value for the C parameter in order to apply regularisation. Note that in this library what we are specifying is the cost value C (importance of the approximation error versus the regularisation error), in such a way that $\eta = \frac{1}{C}$. We will use the `saga solver` and maximum of iterations (`max_iter`) of 10.

3. Experiments

We will test different configurations of the neural network and execute each configuration with five seeds (1, 2, 3, 4 and 5). Based on the results obtained, the average and standard deviation of the error will be obtained. For the regression problems, only the MSE will be shown. However, for classification problems, the CCR (the percentage of correct classified patterns) will be shown. To analyse algorithmic bias in the COMPAS dataset we will use the false positive rate, which is the most appropriate metric for this particular problem, but you can optionally include the false negative rate.

To assess how the implemented algorithm works, we will run it on three different regression datasets:

- *Sin-function dataset*: This dataset is composed of 120 training patterns and 41 testing patterns. It has been obtained by adding some random noise to the sin function (see Figure 1).

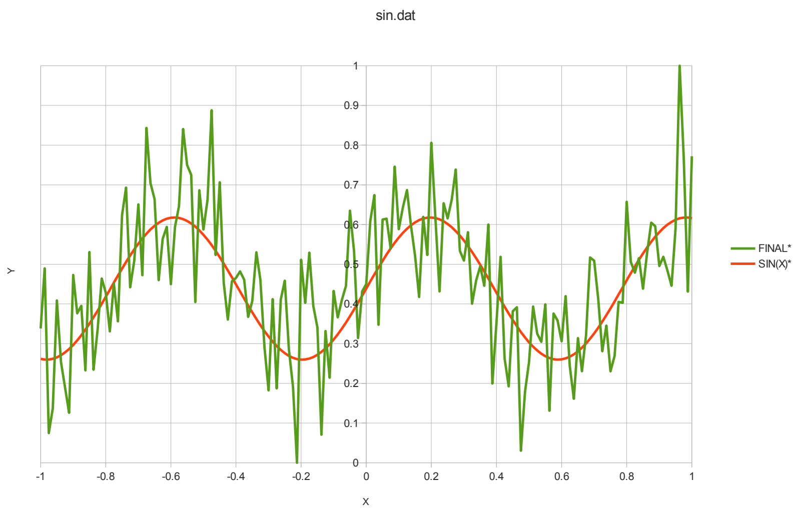

Figura 1: Data representation of the data included in the sin-function estimation problem.

- *Quake dataset*: this dataset is composed by 1633 training patterns and 546 testing patterns. It corresponds to a database in which the objective is to find out the strength of an earthquake (measured on the Richter scale). As input variables, we use the depth of focus, the latitude at which it occurs and the longitude [5].

[5] see `https://sci2s.ugr.es/keel/dataset.php?cod=75` to seek more information.

- *Auto MPG dataset*: The Auto MPG dataset, available in the UCI Machine Learning Repository[6], is a well-known dataset frequently used in machine learning and data analysis tasks. It contains information about various car models, including their attributes and fuel efficiency. This dataset is often used for regression and predictive modeling tasks, particularly to predict a car's miles per gallon (MPG) based on its features (cylinders, displacement, horsepower, etc.). The original dataset has 398 samples and 7 attributes. We binarized discrete attributes and removed model name so the final dataset attributes list is: cylinders, displacement, horsepower, weight, acceleration, modelYear, USA, Europe, Japan and MPG (target variable).

And two classification datasets:

- *ProPublica COMPAS*: This dataset is about the performance of COMPAS algorithm, a statistical method for assigning risk scores within the United States criminal justice system created by Northpointe. It was published by ProPublica in 2016 [7], claiming that this risk tool was biased against African-American individuals (we will deal with this in the following assignment). In this dataset, they analyzed the COMPAS scores for "risk of recidivism" so each individual has a binary "recidivism" outcome, that is the prediction task, indicating whether they were rearrested within two years after the first arrest (the charge described in the data). We reduced the original dataset from 52 to 9 attributes similarly to the original dataset: sex, age, age_cat, juv_fel_count, juv_misd_count, juv_other_count, priors_count, race, c_charge_degree. The prediction variable is whether the individual will be rearrested in two years or not.

  1. sex: binary sex.
  2. age: numerical age.
  3. age_cat: categorical (age < 25, age ≥ 25 and age < 45, age ≥ 45).
  4. juv_fel_count: a continuous variable containing the number of juvenile felonies.
  5. juv_misd_count: a continuous variable containing the number of juvenile misdemeanors.
  6. juv_other_count: a continuous variable containing the number of prior juvenile convictions that are not considered either felonies or misdemeanors.
  7. priors_count: a continuous variable containing the number of prior crimes committed.
  8. race[8]: binary attribute (0 means 'white' and 1 means 'black').
  9. c_charge_degree: Degree of the crime. It is either M (Misdemeanor), F (Felony), or O (not causing jail time).

  This database presents a class imbalance problem for each group, as there are 1361 negative vs 1313 positive labels for black people whereas there are 839 negative vs 544 positive labels for white people. Typically, this translates into models that will tend to label black people as reoffenders whilst it will tend to label white people as not reofenders. This can be assessed though the false positive rate for each group. We will perform an exploratory analysis of this database within the practical sessions. In this lab assignment, the input variables of this dataset have been previously standarized in the `csv` version. The race variable is the last one in the `csv`.

- *noMNIST dataset*: originally, this dataset was composed by $200{,}000$ training patterns and $10{,}000$ test patterns, with a total of 10 classes. Nevertheless, for this lab assignment, the size of the dataset has been reduced in order to reduce the computational cost. In this sense, the dataset is composed by 900 training patterns and 300 test patterns. It includes a set of

[6] Raw dataset and attribute description available here: `https://archive.ics.uci.edu/dataset/9/auto+mpg`
[7] `https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm`
[8] Recommended reading: There's No Scientific Basis for Race–It's a Made-Up Label

letters (from a to f) written with different typologies or symbols. They are adjusted to a squared grid of 28×28 pixels. The images are in grey scale in the interval $[-1{,}0; +1{,}0]$[9]. Each of the pixels is an input variable (with a total of $28 \times 28 = 784$ input variables) and the class corresponds to a written letter (a, b, c, d, e y f, with a total of 6 classes). Figure 2 represents a subset of 180 training patterns, whereas figure 3 represents a subset of 180 letters from the test set. Moreover, all the letters are arranged and available in Moodle in the files `train_img_nomnist.tar.gz` and `test_img_nomnist.tar.gz`, respectively.

Figura 2: Subset of letters belonging to the training dataset.

Figura 3: Subset of letters belonging to the test dataset.

The average and standard deviation of two measures (regression) or four measures (classification) should be computed:

- Regression: average and standard deviation of training and testing MSE.
- Classification: average and standard deviation of training and testing CCR.

At least, the following configurations should be tried:

- *Network architecture*:
  - For all the datasets, consider a number hidden neurons (n_1) equal to the $5\,\%$, $15\,\%$, $25\,\%$ and $50\,\%$ of the total number of patterns of the dataset. In this stage, for classification problems use L1 regularisation and $\eta = 10^{-5}$.
- For the classification problems, once decided the best architecture, try the following values for η: η: $\eta = 10^{-3}$, $\eta = 10^{-2}$, $\eta = 10^{-1}$, $\eta = 1$, $\eta = 10^{1}$, $\eta = 10^{2}$, $\eta = 10^{3}$, along with the two types of regularisation (L2 y L1). What is happening? Compute the difference in number of coefficients for COMPAS and noMNIST dataset when the regularisation type is modified (L2 vs L1)[10].

[9] Check `http://yaroslavvb.blogspot.com.es/2011/09/notmnist-dataset.html` for more information.

[10] The coefficients are in the `coef_` attribute of the logistic regression object. Consider that if the absolute value of a coefficient is lower than 10^{-5}, then the coefficient is null

- Implement the option `-v` so that, in classification problems, the classifier is trained using `sklearn.linear_model.LogisticRegressionCV` instead of using the class `sklearn.linear_model.LogisticRegression`. This will automatically calculate (without looking at the test values), the optimal value of η. Set the object so that the values to be tested (`Cs`) are $\eta = 10^{-3}$, $\eta = 10^{-2}$, $\eta = 10^{-1}$, $\eta = 1$, $\eta = 10^{1}$, $\eta = 10^{2}$ and $\eta = 10^{3}$ and that the optimization is done using a stratified *k-fold* with $k = 3$. Compare, on the two classification datasetss, the results you get using this strategy (under the assumption of L1 and the best ratio of RBFs obtained above). Compare also the values you choose for eta with those obtained by the experimentation of the previous point. What can the differences be due to?

- For one of the classification problems, run the script considering the problem as if it was regression (i.e. the classification parameter is False and compute the CCR rounding the predictions to the closest integer). What is happening for this situation?

As a guideline, the training and generalisation errors achieved by a linear regression (using Weka) over the three regression datasets is shown:

- *sin dataset*: $MSE_{\text{train}} = 0{,}02968729$; $MSE_{\text{test}} = 0{,}03636649$.
- *Quake dataset*: $MSE_{\text{train}} = 0{,}03020644$; $MSE_{\text{test}} = 0{,}02732409$.
- *MPG dataset*: $MSE_{\text{train}} = 0{,}00000$; $MSE_{\text{test}} = 0{,}00750$.

Also, the training CCR and the test CCR achieved by a logistic regression (using Weka) over the two classification datasets is shown:

- *COMPAS dataset*: $CCR_{\text{train}} = 67{,}7348\,\%$; $CCR_{\text{test}} = 66{,}8514\,\%$.
- *noMNIST dataset*: $CCR_{\text{train}} = 80{,}4444\,\%$; $CCR_{\text{test}} = 82{,}6667\,\%$.

The student should be able to improve this error values with some of the configurations.

3.1. File format

The files containing the datasets will be CSV, in such a way that the values will be separated by commas. In this sense, there are no headers. In order to read the files properly, the function `read_csv` from `pandas` should be used. In the COMPAS database, the 'race' variable has been placed in the last column so that it can be easily processed and integrated with `fairlearn`.

4. Deliverables

The files to be submitted will be the following:

- Report in a `pdf` file describing the programme implemented, including results, tables and their analysis.
- Python script.

4.1. Report

The report for this lab assignment must include, at least, the following content:

- Cover with the lab assignment number, its title, subject, degree, faculty department, university, academic year, name, DNI and email of the student
- Index of the content with page numbers.
- Description of the steps for the RBF training stage (**1 page maximum**).

- Experiments and results discussion:
  - Brief description of the datasets used.
  - Brief description of the values of the parameters considered.
  - Results obtained, according to the format specified in the previous section.
  - Discussion/analysis of the results. The analysis must be aimed at justifying the results obtained instead of merely describing the tables. This analysis should include algorithmic bias assessment in COMPAS. Take into account that this part is extremely decisive in the lab assignment qualification. The inclusion of the following comparison items will be appreciated:
    - Test confusion matrix of the best neural network model achieved for the *noMNIST* database. Analysing the errors, **including the images of some letters for which the model mistakes**, to visually check if they are confusing. Comparison between the confusion matrix obtained for this assignment against the one obtained in the previous lab assignment.
    - Computational time needed for the training step for *noMNIST* dataset and comparison against the computational time spent in the previous lab assignment.
- Bibliographic references or any other material consulted in order to carry out the lab assignment different to the one provided by the lecturers (if any).

Although the content is important, the presentation, including the style and structure of the document will also be valued. The presence of too many spelling mistakes can decrease the grade obtained.

4.2. Executable and source code

Together with the report, the script file prepared to be run in the UCO's machines (concretely, test using `ssh` on `ts.uco.es`) must be included. In addition, all the source code must be included. The script developed should receive the following command-line arguments[11]:

- Argument `-t`, `--train_file`: Indicates the name of the file that contains the training data to be used. This argument is compulsory, and without it, the program can not work.
- Argument `-T`, `--test_file`: Indicates the name of the file that contains the testing data to be used. If it is not specified, training data will be used as testing data.
- Argument `-s`, `--standarize`: Boolean indicating whether the data sets are to be standardized (inputs and outputs for regression, only inputs for classification). If not specified, we will assume that it is not standardized.
- Argument `-c`, `--classification`: Boolean that indicates whether it is a classification problem. If it is not specified, we will suppose that it is a regression problem.
- Argument `-r`, `--ratio_rbf`: Indicates the radium (by one) of RBF neurons with respect to the total number of patterns in training. If not specified, use $0{,}1$.
- Argument `-l`, `--l2`: Boolean that indicated if L2 regularisation is used, instead of L1. If it is not specified, L1 will be used.
- Argument `-e`, `--eta`: Indicates the value for the eta (η) parameter. By default, use $\eta = 1e-2$.
- Argument `-f`, `--fairness`: Boolean that indicated if fairness metrics should be extracted from predictions. Assumes that the group is stored as the last variable of the input variables. By default, it is disabled.

[11] To process the input sequence, the `click` library will be used.

- Argument `-o`, `--outputs`: Indicates the number of output columns of the dataset (always placed at the end). By default, use $o = 1$.
- Argument `-v`, `--logisticcv`: Boolean for activating the use of the class `LogisticRegressionCV` for classification problems. By default, it is assumed that it will not be applied (therefore, `LogisticRegression` is used).
- (Kaggle) Argument `-p`, `--pred`: Boolean that indicates if the prediction mode is used.
- (Kaggle) Argument `-m`, `--model_file`: Indicates the directory in which the trained models are saved (in the training mode, without the flag p) or the file containing the model that will be used (in the prediction mode, with the flag p).
- Argument `--help`: It shows the help of the program (use the one automatically generated by the `click` library)

An example of execution can be seen in the following output[12]:

```
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]
-s, --standarize       Standarize input variables (and outputs if it is a
regression problem).  [default: False]
-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]
-f, --fairness         Evaluates prediction using fairlern metrics. It is
assumed that last input variable is the group
variable.  [default: False]
-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]
-v, --logisticcv       Consider LogisticRegressionCV.  [default: False]
-m, --model TEXT       Directory name to save the models (or name of the
file to load the model, if the prediction mode is
active).
--help                 Show this message and exit.

# Example with COMPAS dataset and fairness parameter
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./csv/train_compas.csv -T ./csv/
    test_compas.csv -c --l2 -f
-----------
Seed: 1
-----------
Number of RBFs used: 405
```

[12]To make the developed code to work in the UCO machines, the packages `click` and the last version of the package `scikit-learn` should be installed, using the following commands:
`pip install scikit-learn --user --upgrade`
`pip install click --user --upgrade`

```
/home/javi/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350:
    ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
Training MSE: 0.204360
Test MSE: 0.212523
Training CCR: 68.20 %
Test CCR: 66.69 %
-----------
Seed: 2
-----------
Number of RBFs used: 405
/home/javi/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350:
    ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
Training MSE: 0.204601
Test MSE: 0.212833
Training CCR: 68.23 %
Test CCR: 66.69 %
-----------
Seed: 3
-----------
Number of RBFs used: 405
/home/javi/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350:
    ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
Training MSE: 0.204681
Test MSE: 0.212844
Training CCR: 68.01 %
Test CCR: 66.85 %
-----------
Seed: 4
-----------
Number of RBFs used: 405
/home/javi/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350:
    ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
Training MSE: 0.204666
Test MSE: 0.213130
Training CCR: 67.93 %
Test CCR: 66.69 %
-----------
Seed: 5
-----------
Number of RBFs used: 405
/home/javi/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:350:
    ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
Training MSE: 0.204510
Test MSE: 0.212759
Training CCR: 68.20 %
Test CCR: 66.69 %
******************
Summary of results
******************
Training MSE: 0.204564 +- 0.000118
Test MSE: 0.212818 +- 0.000194
Training CCR: 68.11 % +- 0.12 %
Test CCR: 66.72 % +- 0.07 %
Training FN0: 59.85 % +- 0.34 %
Training FN1: 32.55 % +- 0.27 %
Test FN0: 51.55 % +- 0.85 %
Test FN1: 37.66 % +- 0.31 %
Training FP0: 14.11 % +- 0.66 %
Training FP1: 31.02 % +- 0.16 %
Test FP0: 21.59 % +- 0.31 %
Test FP1: 28.89 % +- 0.29 %

```

```
# En los siguientes ejemplos, los CCRs salen 0 porque es un problema de regresión
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./train_mpg.csv -T ./test_mpg.csv -r 0.5
    -s
-----------
Seed: 1
-----------
Number of RBFs used: 156
Training MSE: 0.037012
Test MSE: 0.277343
Training CCR: 0.00%
Test CCR: 0.00%
-----------
Seed: 2
-----------
Number of RBFs used: 156
Training MSE: 0.040975
Test MSE: 0.140427
Training CCR: 0.00%
Test CCR: 0.00%
-----------
Seed: 3
-----------
Number of RBFs used: 156
Training MSE: 0.037219
Test MSE: 0.182746
Training CCR: 0.00%
Test CCR: 0.00%
-----------
Seed: 4
-----------
Number of RBFs used: 156
Training MSE: 0.044984
Test MSE: 0.145340
Training CCR: 0.00%
Test CCR: 0.00%
-----------
Seed: 5
-----------
Number of RBFs used: 156
Training MSE: 0.040233
Test MSE: 0.174549
Training CCR: 0.00%
Test CCR: 0.00%
******************
Summary of results
******************
Training MSE: 0.040084 +- 0.002914
Test MSE: 0.184081 +- 0.049390
Training CCR: 0.00% +- 0.00%
Test CCR: 0.00% +- 0.00%

i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./train_mpg.csv -T ./test_mpg.csv -r
    0.15 -s
-----------
Seed: 1
-----------
Number of RBFs used: 46
Training MSE: 0.087675
Test MSE: 0.101936
Training CCR: 0.00%
Test CCR: 0.00%
-----------
Seed: 2
-----------
Number of RBFs used: 46
Training MSE: 0.105456
```

```
Test MSE: 0.127247
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 3
-----------
Number of RBFs used: 46
Training MSE: 0.086466
Test MSE: 0.107544
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 4
-----------
Number of RBFs used: 46
Training MSE: 0.086680
Test MSE: 0.095453
Training CCR: 0.00 %
Test CCR: 0.00 %
-----------
Seed: 5
-----------
Number of RBFs used: 46
Training MSE: 0.089423
Test MSE: 0.116676
Training CCR: 0.00 %
Test CCR: 0.00 %
******************
Summary of results
******************
Training MSE: 0.091140 +- 0.007234
Test MSE: 0.109771 +- 0.011175
Training CCR: 0.00 % +- 0.00 %
Test CCR: 0.00 % +- 0.00 %

i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t ./train_sin.csv -T ./test_sin.csv -r
    0.15 -o 1
-----------
Seed: 1
-----------
Number of RBFs used: 18
Training MSE: 0.012100
Test MSE: 0.104193
Training CCR: 0.83 %
Test CCR: 2.44 %
-----------
Seed: 2
-----------
Number of RBFs used: 18
Training MSE: 0.011399
Test MSE: 0.200716
Training CCR: 0.83 %
Test CCR: 2.44 %
-----------
Seed: 3
-----------
Number of RBFs used: 18
Training MSE: 0.011953
Test MSE: 0.102114
Training CCR: 0.83 %
Test CCR: 2.44 %
-----------
Seed: 4
-----------
Number of RBFs used: 18
Training MSE: 0.012089
Test MSE: 0.082532
```

```
Training CCR: 0.83%
Test CCR: 2.44%
-----------
Seed: 5
-----------
Number of RBFs used: 18
Training MSE: 0.011961
Test MSE: 0.092528
Training CCR: 0.83%
Test CCR: 2.44%
******************
Summary of results
******************
Training MSE: 0.011901 +- 0.000258
Test MSE: 0.116417 +- 0.042847
Training CCR: 0.83% +- 0.00%
Test CCR: 2.44% +- 0.00%
```

4.3. [OPTIONAL] Save the model to a file.

During the training stage, the script can save the model trained as a pickle[13]. This will allow to use the trained model to predict the outputs of the **Kaggle** dataset.

To save the model, it is necessary to use the `-m` parameter. An execution example is as follows:

```
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -t train.csv -T val.csv -l -c -r 0.1 -m
    model
-----------
Seed: 1
-----------
Number of RBFs used: 157
Training MSE: 0.121783
Test MSE: 0.140203
Training CCR: 82.60%
Test CCR: 78.67%
-----------
Seed: 2
-----------
Number of RBFs used: 157
Training MSE: 0.122003
Test MSE: 0.139427
Training CCR: 82.35%
Test CCR: 79.05%
-----------
Seed: 3
-----------
Number of RBFs used: 157
Training MSE: 0.123603
Test MSE: 0.138931
Training CCR: 82.35%
Test CCR: 78.86%
-----------
Seed: 4
-----------
Number of RBFs used: 157
Training MSE: 0.122647
Test MSE: 0.138973
Training CCR: 82.16%
Test CCR: 78.67%
-----------
Seed: 5
-----------
Number of RBFs used: 157
Training MSE: 0.124017
```

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

```
Test MSE: 0.142853
Training CCR: 82.48%
Test CCR: 78.67%
******************
Summary of results
******************
Training MSE: 0.122811 +- 0.000874
Test MSE: 0.140077 +- 0.001461
Training CCR: 82.39% +- 0.15%
Test CCR: 78.78% +- 0.15%
```

Once the execution is finished, there will be a folder named "model" containing 5 pickles. Each one corresponds with the generated model for each seed. In order to obtain the predictions, one of these 5 pickles should be chosen.

```
i02gupep@NEWTS:~/imc/workspace/la3$ ls model/
 1.pickle  2.pickle  3.pickle  4.pickle  5.pickle
```

4.4. [OPTIONAL] Obtaining the predictions for Kaggle.

Once the model is saved to a pickle, it is possible to obtain the output predictions for the Kaggle dataset. For this, `-m` and `-p` parameters should be used. Below is an example:

```
i02gupep@NEWTS:~/imc/workspace/la3$ ./rbf.py -T test.csv -p -m model/2.pickle
Id,Category
0,1
1,0
2,0
3,0
4,0
5,0
6,1
7,0
8,0
9,0
10,0
11,0
12,1
13,0
14,1
15,0
16,0
17,1
18,0
19,0
20,0
21,0


...

1887,0
1888,0
1889,0
1890,1
1891,1
1892,0
1893,0
1894,0
1895,1
1896,0
1897,0
1898,1
1899,0
```

The output can be redirected to a `csv` file:

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

This file is ready to be uploaded to Kaggle.