Programming Exercise 7: Tyreoid Desease Detection using Radial Basis Function and Probabilistic Neural Networks

Thiago Raulino Dal Pont
Automation and Systems Department
Federal University of Santa Catarina

July 10, 2022

1 Introduction

In this seventh programming exercise, we are going to focus on the tireoid classification problem based on Radial Basis Function (RBF) Networks and Probabilistic Neural Networks (PNN).

Our goal is to maximize the accuracy on the test set. To do so, based on the Keras, Scikit Learn libraries, and other libraries available on Github for RBF and PNN, we implemented and train models to make predictions using the dataset from the University of California (the same used in the previous assigment).

This work is divided as follows:

- Section 2 presents the required materials, methods and studies to finish the exercise;
- Section 3 presents the results and discussions.

2 Methods and Materials

For model evaluation, we applied accuracy to estimate how well the models fit the data:

$$Acc = 100 \cdot \frac{|y_{act} - y_{pred}|}{y_{act}} \tag{1}$$

In terms of tools versions, we used:

- Python 3.9.12
- Numpy 1.21.5
- Matplotlib 3.5.1
- Scipy 1.7.3
- Keras 2.4.3

Regarding the RBF implementation, we used a library that implements a RBF layer for Keras [1]. And for PNN, we used a simple code available on Github [2]. In the former, we just apply the layer as usual, however is the latter, we had to make some adjustments in the data to fit the code and avoid extra and unnecessary effort.

3 Experiments

In this section, we present the experiments applied to the RBF and PNN.

Figure 1: RBF network

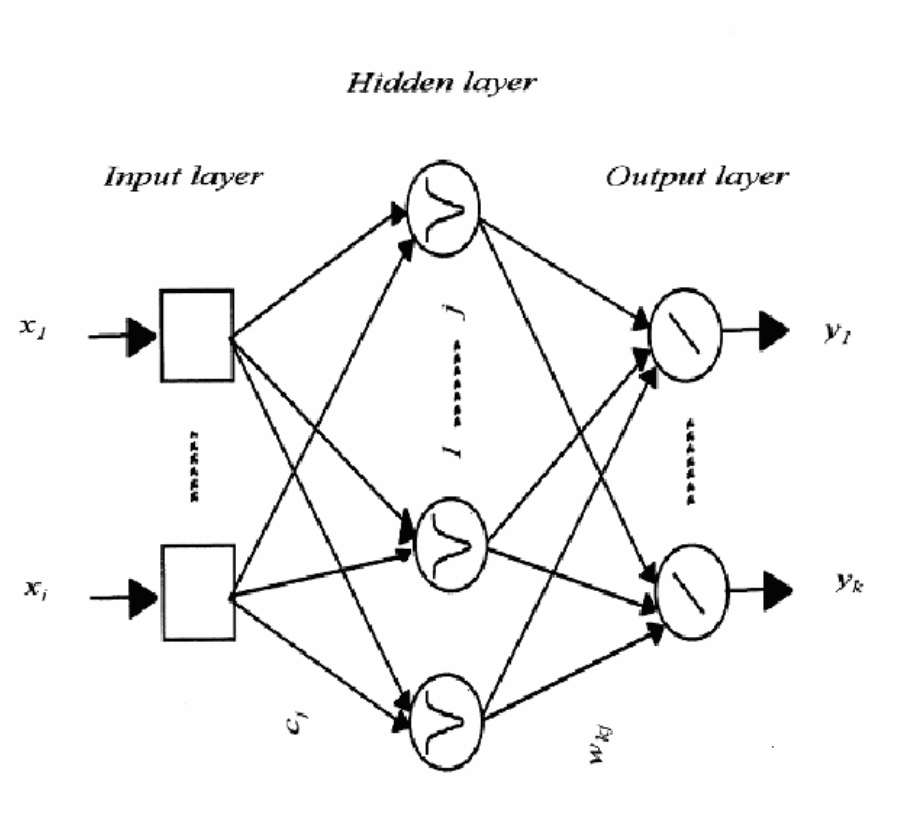


Table 1: RBF network hyperparameters

Hyperparameter	Value
Epochs	200
Batch size	32
Optimizer	Adam
Learning Rate	0.001
Hidden L Size	100
Output Activation	Softmax
Beta	2.0

3.1 RBF experiments

Regarding RBFs, we designed the network according to the Figure 1.

In terms of hyperparameters, after some tests, we selected those presented in Table 1.

After training the RBF model, and using it to make predictions in the test set, it achieved the results presented in Figure 2. One can note that RBF could satisfactorily learn the three classes resulting a macro recall of 0.95 and F1-Score of 0.82.

The confusion matrix for RBF is shown in Figure 3. One can see that predicted 59 patients with the disease when the patient actually did not have it (False positives). And predicted 4 patients without disease when they actually had them (False negatives). Thus, considering we focus on avoiding false negatives, we can state that the RBF model is a good predictor for this problem.

3.2 PNN experiments

In this set of experiments, we applied a PNN to the same dataset. We used the default hyperparameters from the implementation in [2].

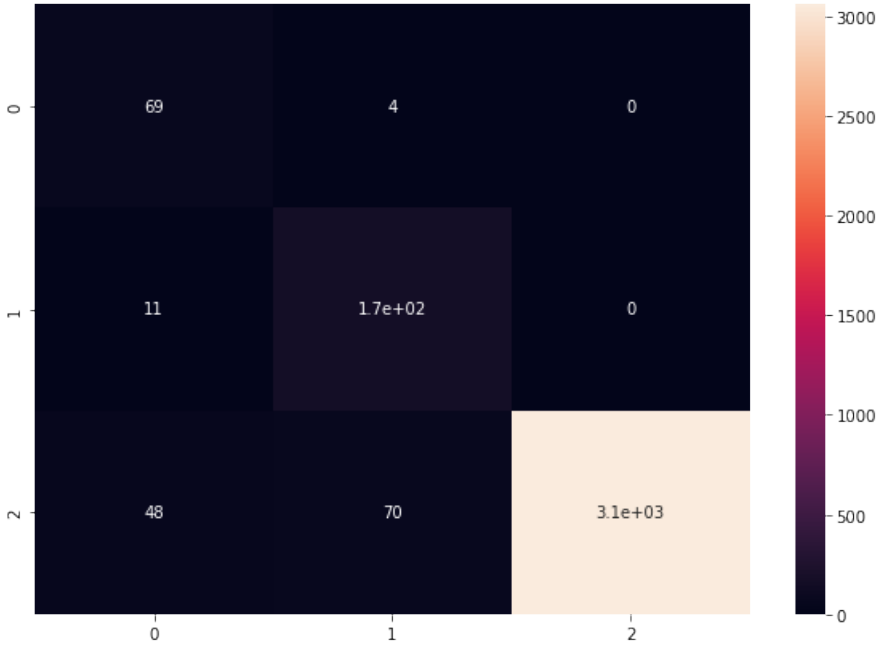
The results are shown in Figure 4. One can note the poor results achieved by the PNN.

The confusion matrix for PNN is shown in Figure 3. One can see that the PNN predict that all patients do not have any disease. Thus, this is not a good model for this problem.

Figure 2: Results for RBF in the test set

	precision	recall	f1-score	support
0	0.54	0.95	0.69	73
1	0.69	0.94	0.80	177
2	1.00	0.96	0.98	3178
accuracy			0.96	3428
macro avg	0.74	0.95	0.82	3428
weighted avg	0.97	0.96	0.97	3428

Figure 3: Confusion Matrix for RBF in the test set



References

[1] "RBF-keras: an rbf layer for keras library." https://github.com/PetraVidnerova/rbf_keras. Accessed 07 Jul 2022.

[2] "Probabilistic neural network in python." https://github.com/JaeDukSeo/probabilistic-neural-network-in-python. Accessed 07 Jul 2022.

Figure 4: Results for PNN in the test set

	precision	recall	f1-score	support
0	0.02	1.00	0.04	73
1	0.00	0.00	0.00	177
2	0.00	0.00	0.00	3178
accuracy			0.02	3428
macro avg	0.01	0.33	0.01	3428
weighted avg	0.00	0.02	0.00	3428

Figure 5: Confusion Matrix for PNN in the test set