Radial Basis Function for Boundary Detection

December 14, 2025

Say, we are faced with a boundary detection problem. For simplicity, lets only have two classes – reds and blues. The true boundary, which is known from the construction, is shown in the figure below. Its got quite a bit of complexity to it. In fact, a handful of samples are mislabeled as well, to mimic the situation in the real world.

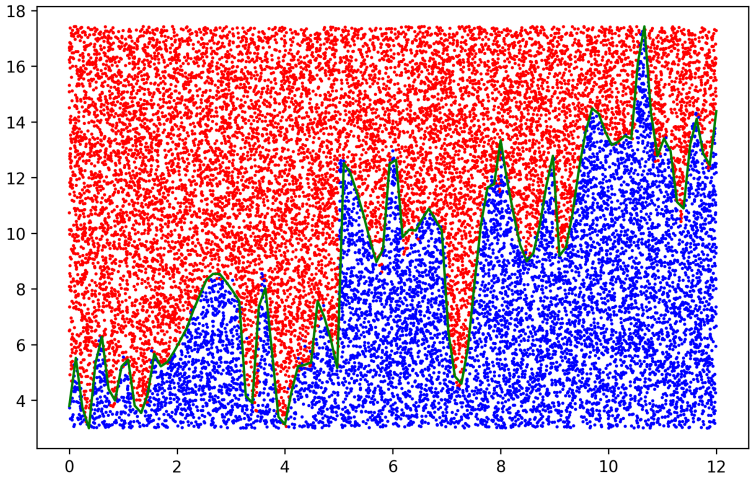

Figure 1: True boundary and samples

Note:

Turning a blind eye to the true boundary, we make it disappear. So what we have are red and blue dots and quite noisy spikes. The job is to estimate the function governing the boundary.

Given the complexity of the problem, we would want to extract more information from the two features, x and y, before building a model. In other words, we are systematically preprocessing the raw inputs before feeding

them to the model to make the job easier for the model. This is when radial basis function come in.

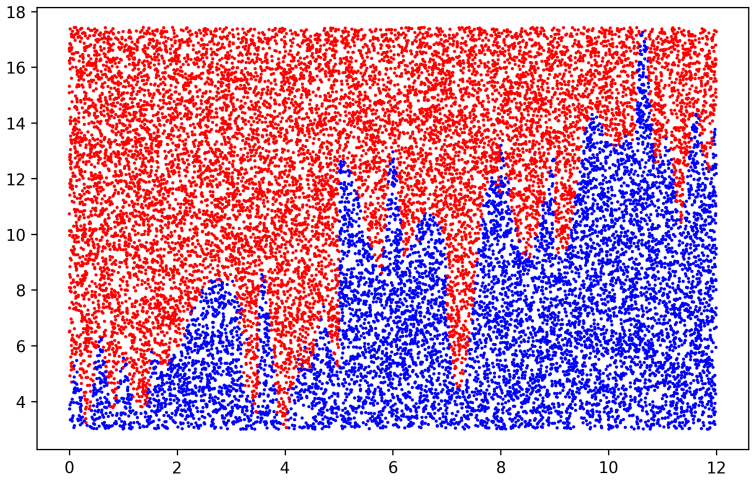

Figure 2: Samples

Note:

1 Setup and intuition

We have information on $(X_n,\ Y_n) \in D,\ n = \{1, ..., N\}$, which influences the true function $h(x)$. X_n represents features $(x_n,\ y_n)$, whereas Y_n consist of targets. Given the complexity of $h(x)$, we may be asking a lot from even deep learning models when using raw inputs as features. What if we could extract localized information from the raw inputs that can complement the model?

Let's think of a similarity measure, which can help identify how similar or dissimilar the two given points are. Simply, we can consider the distance function: $||X - X_n^r||^2$, where X consists of data or sample points (raw iputs) and X_n^r is the local reference point. In our case, this is simply going to be the Eucledian distance between the two points. In case of the sample point X_1, we'll have:

$$(\mathrm{x}_1 - x_n^r)^2 + (y_1 - y_n^r)^2 \quad (1)$$

where, $(x_1,\ y_1)$ is the sample point and $(x_n^r,\ y_n^r)$ is the reference point. If we are to just take the sample point $(x_1,\ y_1)$ and calculate similarity across all points $(n = \{1, ..., N\})$, then we'll get similarly measure of $(x_1,\ y_1)$ with respect to all sample points. Points that are closer to $(x_1,\ y_1)$ will have lower magnitude, where are points that are far away will have higher magnitude. Mathematically, we'll have a $1 \times N$ matrix as given below:

$$[||X_1 - X_1||^2\ \ ||X_1 - X2||^2\ \ ...\ \ ||X_1 - X_N||^2]$$

We'd want to implement this process for all sample points. Once we've done that we'll have a $N \times N$ matrix as given below:

$$\mathbf{S} = \begin{bmatrix} ||X_1 - X_1||^2 & ||X_1 - X_2||^2 & \cdots & ||X_1 - X_N||^2 \\ ||X_2 - X_1||^2 & ||X_2 - X_2||^2 & \cdots & ||X_2 - X_N||^2 \\ \vdots & \vdots & \ddots & \vdots \\ ||X_N - X_1||^2 & ||X_N - X_2||^2 & \cdots & ||X_N - X_N||^2 \end{bmatrix} \tag{2}$$

The diagonal entries will be zero. The matrix above is the radial matrix extracted from radial $||X - X_n||$, where X_n is the center point. So far, we know that smaller measures in the matrix signals that the two points are closer. Next, we'd want to feed this information into a basis function. For this, we'll choose a Gaussian function given as:

$$exp(-\gamma||X - X_n||^2) \tag{3}$$

Let's think through equation 3 carefully by fixing $\gamma = 1$ for simplicity. Points that are closer will now take a higher measure, where as points far apart will have measures lower in magnitude. If the two points (sample and center) are way too far away, the Gaussian basis function will just scale such points to 0. This now signifies the entire basis we started out with – closer or points have higher measure index, while points further apart will have lower measure.

Next, γ is the parameter that controls the "locality" or "bandwidth" of influence. Let's consider two points $X_1 = (x_1, y_1)$ and $X_2 = (x_2, y_2)$. Say, the distance measures of X_1 and X_2 from a given center X_n are 0.5 and 1.5 respectively (these are the radial values). When $\gamma = 1$, the radial

basis function values from the Gaussian kernel are 0.607 and 0.223 for the respective points. But now consider $\gamma = 10$. The radial basis function values become 0.0067 and close to 0 for the two points. Thus, larger values of γ enforce strict locality—only nearby points significantly influence $h(x)$, while even moderately distant points are heavily penalized. In this sense, $\exp(-\gamma||X - X_n||)$ is the Gaussian RBF kernel, where γ dictates how quickly the influence decays with distance. The Gaussian kernel provides a smooth, continuous measure of influence. Unlike hard threshold methods (e.g., k-nearest neighbors), where points are either "in" or "out," the Gaussian allows influence to decay gradually. This smoothness will be crucial for gradient-based learning in neural networks.

After inputting the radial values from $\mathbf{S}$, given in 2, into the Gaussian basis function, we'll have the following matrix Φ:

$$\Phi = exp(-\gamma \times \mathbf{S}) \tag{4}$$

After this, we are prepared to look at the RBF model.

2 Model

The model is written as:

$$h(x) = \sum_{n=1}^{N} W_n \times exp(-\gamma||X - X_n||^2) \; for \; D = (x_1,\; y_1), ..., (x_n,\; y_n) \tag{5}$$

W_n are the weights that are adjusted so that the LHS=RHS. We would want to conduct inference for every point. Here, we have N data points, N parameters, and N equation. Hence, we can perfectly identify W_n so that the in sample error is 0. Let's express equation 5 in a matrix form.

$$\Phi \cdot W = h(x) \tag{6}$$

where, Φ is the radial basis function matrix given explicitly as:

$$\mathbf{\Phi} = \begin{bmatrix} exp(-\gamma||X_1 - X_1||^2) & exp(-\gamma||X_1 - X_2||^2) & \cdots & exp(-\gamma||X_1 - X_N||^2) \\ exp(-\gamma||X_2 - X_1||^2) & exp(-\gamma||X_2 - X_2||^2) & \cdots & exp(-\gamma||X_2 - X_N||^2) \\ \vdots & \vdots & \ddots & \vdots \\ exp(-\gamma||X_N - X_1||^2) & exp(-\gamma||X_N - X_2||^2) & \cdots & exp(-\gamma||X_N - X_N||^2) \end{bmatrix} \tag{7}$$

The solution to 6 requires that matrix Φ is invertible. It's solution is given as:

$$\boxed{W = \Phi^{-1} \cdot h(x)} \tag{8}$$

Note that equation 8 gives the exact solution of the weight matrix W that leads to an insample error of 0. However, the model won't be generalizable as it's out of sample predictive capacity won't be any good.

To improve the generalizable property of the model, we would want to reduce the number of parameters to be estimated to K such that $K << N$. Surely this will induce some in sample error, but we are not too concerned about that. Want we want is a model that is generalizable. To do so, rather than creating the radial basis features in relation to each data point, we'd want to limit the number of centers to K. Re-writing equation 5, we have the following.

$$h(x) = \sum_{k=1}^{K} W_k \times exp(-\gamma ||X - \mu_k||^2) \; for \; D = (x_1,\; y_1), ..., (x_n,\; y_n) \tag{9}$$

The unknown parameters are $W_k s$ and $\mu_k s$, which presents us with $2K$ parameters to be estimated. This might still seem like a heafty number of parameters. Now, the trick is to estimate μ_ks without touching the labels or $Y s$ using unsupervised ML. In this way, we won't be contaminating the training set. This problem is NP-hard and we'll be using Lloyd's algorithm (K-means) to come up with μ_ks.

2.1 Lloyd's algorithm (K-means)

The objective is to delineate $X_n;\; n = \{1, ..., N\}$ to cluster $\mathcal{S}_k; k = \{1, ..., K\}$ so that the following equation is minimized:

$$\min \sum_{k=1}^{K} \underbrace{\sum_{X \in \mathcal{S}_k} ||X - \mu_k||^2}_{A} \; w.r.t \; \mu_k \; \& \; \mathcal{S}_k \tag{10}$$

The term A shows the closeness measure between X_n ($X \in \mathcal{S}_k$) and μ_k (the center of cluster $\mathcal{S}_k$, i.e., mean of $X \in \mathcal{S}_k$). We'd want to do this for all k, so that μ_k and $\mathcal{S}_k$ minimizes the term above. From a brute force

approach, this would mean assigning N data points into K clusters and exhausting all combination of assignment. For each combination, we'd want to compute the optimal centers (i.e., mean of observations within a cluster). Finally, we'd want to pick the combination that minimizes the closeness. As previously mentioned, this approach is NP-hard. Instead we'll use Lloyd's algorithm to compute the optimal centers.

Lloyd's algorithm

1. Get the inputs X, $\{(x_1,\ y_1)...,(x_n,\ y_n)\}$. We'll only be using the inputs and won't touch Y, the targets.
2. Initialize the centers. We can pick K centers at random from the N inputs.
3. Compute distance from each point to all centers: $||X_n - \mu_l||^2$, where l are the centers $l \in \{1, ..., K\}$ and $n \in \{1, ..., N\}$.
4. For each n, pick μ_k that minimizes the distance: $k = \arg\min_l ||X_n - \mu_l||^2$. This gives the new cluster assignment.
5. Recompute the cluster centers as the mean of all points using the newly assigned cluster from Step 4.
6. Reiterate Steps 4 and 5 until convergence is attained.

The initial cluster means and those after convergence are shown below. We see that the $K = 50$ cluster means are more uniformly spread after convergence.

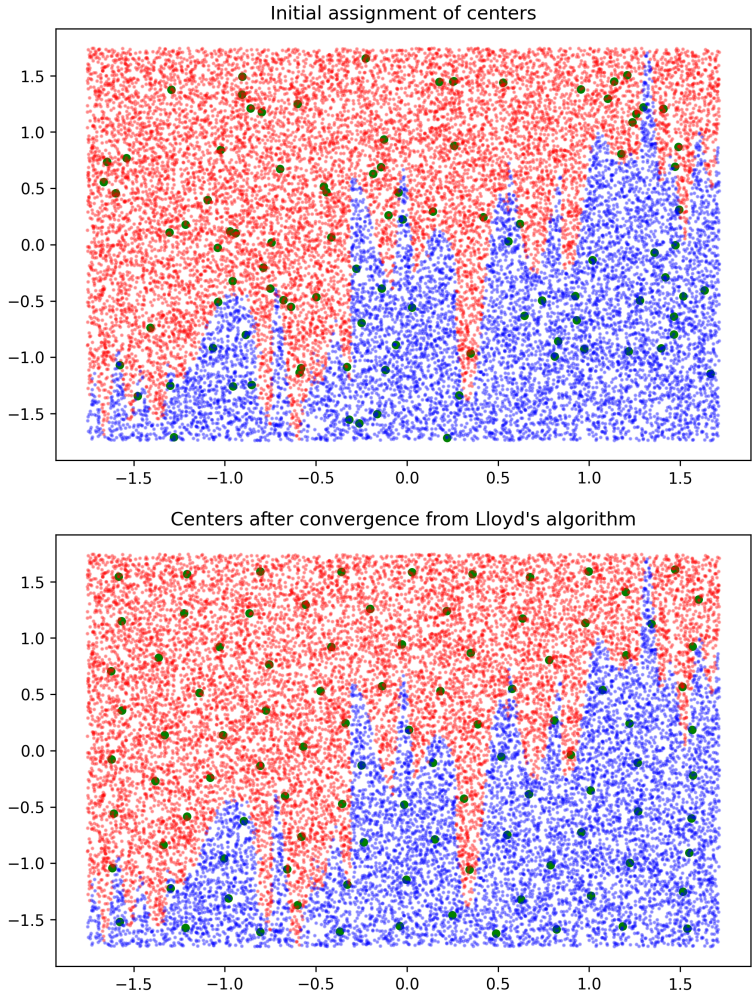

Figure 3: Initial vs. cluster means after convergence

Note:

Using the new center means $\mu_1, ..., \mu_K$, we form the radial basis matrix given as:

$$\bar{\bar{\mathbf{\Phi}}} = \begin{bmatrix} exp(-\gamma||X_1 - \mu_1||^2) & exp(-\gamma||X_1 - \mu_2||^2) & \cdots & exp(-\gamma||X_1 - \mu_K||^2) \\ exp(-\gamma||X_2 - \mu_1||^2) & exp(-\gamma||X_2 - \mu_2||^2) & \cdots & exp(-\gamma||X_2 - \mu_K||^2) \\ \vdots & \vdots & \ddots & \vdots \\ exp(-\gamma||X_N - \mu_1||^2) & exp(-\gamma||X_N - \mu_2||^2) & \cdots & exp(-\gamma||X_N - \mu_K||^2) \end{bmatrix} \tag{11}$$

where,

$$\begin{bmatrix} exp(-\gamma||X_1 - \mu_1||^2) & exp(-\gamma||X_1 - \mu_2||^2) & \cdots & exp(-\gamma||X_1 - \mu_N||^2) \end{bmatrix} \tag{12}$$

are the basis for n = 1.

2.2 Using Neural Net (NN) to find weights

Now that we have the radial basis function matrix, we can use neural net to compute weights, W. Once we train the model using NN, we can evaluate the model performance of the best (final) model in the testing set and use the best (final) model to classify a new set of grid points. Of course, this requires attention to a whole lot of details and the reader is referred to the code, which provides transperency on building of the model.

The performance metrics (accuracy and binary cross entropy loss) are given in the figures below for both training and validation sets. In figure 4, we see that the training accuracy improves and reaches 1 as the model is trained for longer. However, the validation loss decreases successively for a while and starts to show increasing trend later on. This means that as the model is trained for longer epochs it starts recognizing attributes specific to the training set, and it likely won't be good for generability purposes. Hence, we pick a sweet spot and train the final model for 60 epochs using the whole of training set. Then we use this model fit on the testing set to obtain performance metrics. This simple model is able to generate an accuracy of 0.9745 on the test data with a loss of 0.0617. Given the simplicity of the model, these metrics are quite reassuring.

Next, we generate a new grid of features and use the final model to classify this data. This gives us the boundary plot as shown in Figure 5.

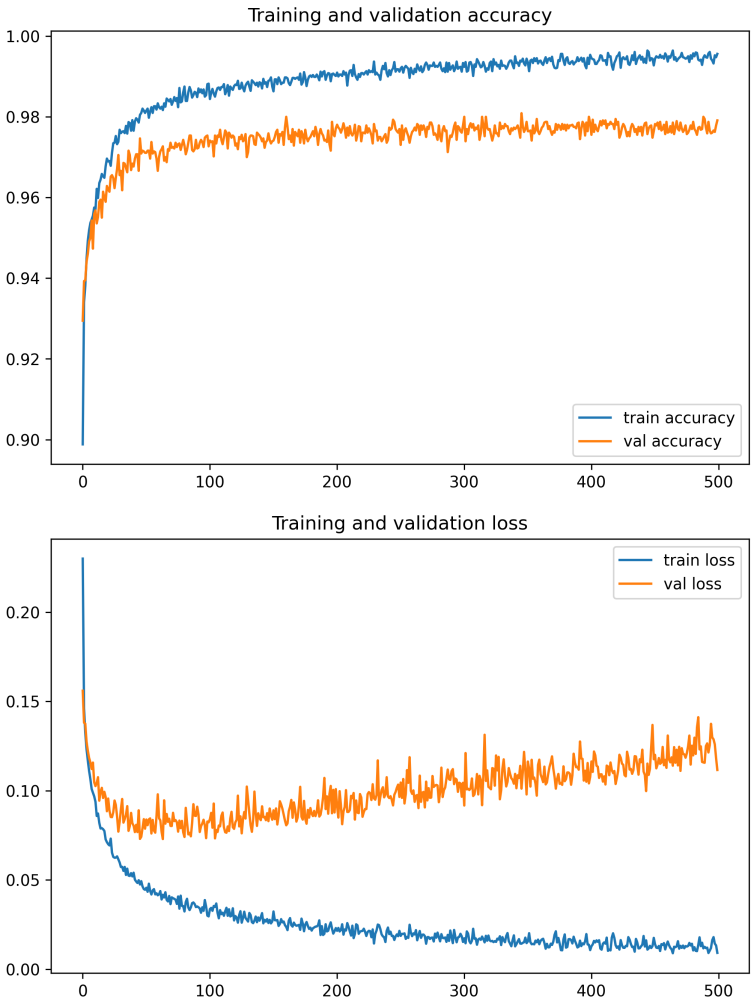

Figure 4: Model performance metrics as a function of epochs

Note:

Our approach of integrating radial basis function with NN is able to capture most of the spikes – sure it misses out on a couple. But as a whole, its not too bad!

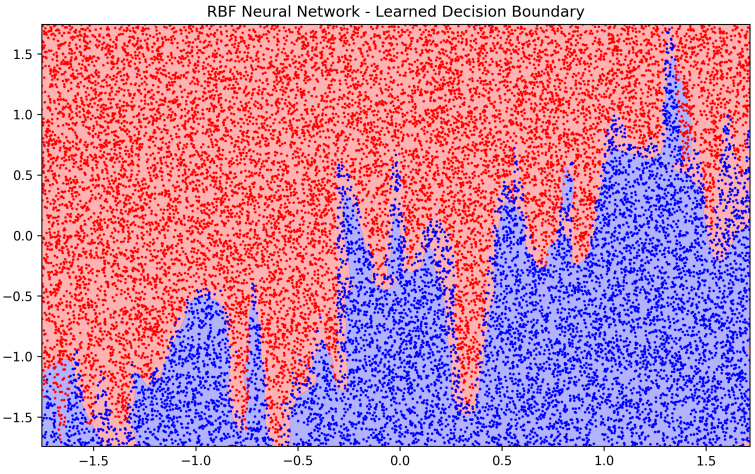

Figure 5: Boundary identified by the neural net model

Note: