A TRAINING METHOD FOR RBF NEURAL NETWORKS USING HIERARCHICAL CLUSTERING

MARIO FERNÁNDEZ SIMÓN

Thesis supervisor
LUIS ANTONIO BELANCHE MUÑOZ (Department of Computer Science)

Degree
Bachelor's Degree in Informatics Engineering (Computing)

Bachelor's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Acknowledgements

I want to thank my parents and friends for always supporting and encouraging me.

I also thank my thesis supervisor, Luís Antonio Belanche Muñoz, for his clear guidance and help.

Abstract

This project focuses on the design, implementation and evaluation of a new training method for Radial Basis Function Neural Networks (RBFNNs). The training process is divided into two stages. The first determines the optimal number of neurons through hierarchical clustering and places them using K-means. The second stage computes the weights with different regularisation techniques: Ridge, Lasso, AIC, and two sequential combinations (Lasso + Ridge and AIC + Ridge).

An exhaustive experimental framework is presented for regression tasks, employingg synthetic data sets—simulating univariate functions with varied noise levels—and real-world data. Each experiment compares the regularisation methods across several evaluation metrics. The performance of our neuron-selection procedure is also benchmarked against the most commonly used method, K-means. Finally, a scalability test with large data sets is conducted.

Experimental results confirm the effectiveness of RBFNNs, delivering accurate and efficient estimates. Comparative analysis shows that the sequential AIC + Ridge model achieves near-state-of-the-art performance at a greatly reduced computational cost relative to more complex models in the literature. Moreover, the scalability test demonstrates that the model retains respectable accuracy for its computational expense, making it a solid choice for large data sets that can run comfortably on low-power hardware.

Keywords: Radial Basis Function Neural Networks, hierarchical clustering, regularisation, computational efficiency.

Resumen

Este proyecto se centra en el diseño, la implementación y la evaluación de un nuevo método de entrenamiento para Redes Neuronales de Función de Base Radial (RBFNN). El proceso de entrenamiento se divide en dos bloques. El primero se encarga de buscar el número óptimo de neuronas usando clustering jerárquico y su colocación usando K-means. El segundo calcula los pesos usando diferentes técnicas de regularización. Los métodos de regularización usados son: Ridge, Lasso, AIC, y dos combinaciones secuenciales (Lasso + Ridge y AIC + Ridge).

El trabajo incluye un marco experimental exhaustivo sobre tareas de regresión usando tanto conjuntos de datos sintéticos que simulan funciones de una variable con diferentes niveles de ruido como conjuntos de datos de la vida real. Para cada experimento se realiza una comparativa entre los diferentes métodos de regularización observando las diferencias para diversas métricas de evaluación. Además, se compara el rendimiento de nuestro método para seleccionar neuronas con el método comúnmente más usado, K-means. Por último, se realiza una prueba de escalabilidad con grandes conjuntos de datos.

La evaluación experimental confirma la eficacia de los modelos RBFNN, proporcionando estimaciones precisas y eficientes. El análisis comparativo revela que el modelo secuencial AIC + Ridge obtiene un rendimiento cercano al estado del arte con un costo computacional muy reducido en comparación con modelos más complejos en literatura científica. Por otro lado, la prueba de escalabilidad muestra que el modelo alcanza una precisión respetable para el costo computacional que conlleva, lo que lo convierte en una opción sólida para grandes conjuntos de datos que puede ejecutarse cómodamente en hardware de baja potencia.

Palabras clave: Radial Basis Function Neural Networks, clustering jerárquico, regularización, eficiencia computacional.

Resum

Aquest projecte se centra en el disseny, la implementació i l'avaluació d'un nou mètode d'entrenament per a Xarxes Neuronals de Funció de Base Radial (RBFNN). El procés d'entrenament es divideix en dos blocs. El primer determina el nombre òptim de neurones mitjançant agrupament jeràrquic i en fixa la posició amb K-means. El segon bloc calcula els pesos amb diferents tècniques de regularització: Ridge, Lasso, AIC i dues combinacions seqüencials (Lasso + Ridge i AIC + Ridge).

El treball presenta un marc experimental exhaustiu per a tasques de regressió, utilitzant tant conjunts de dades sintètics, que simulen funcions d'una variant amb diversos nivells de soroll, com conjunts de dades reals. Cada experiment compara els mètodes de regularització segons diverses mètriques d'avaluació. També es contrasta el rendiment del nostre procediment de selecció de neurones amb el mètode més habitual, K-means. Finalment, es realitza una prova d'escalabilitat amb conjunts de dades grans.

Els resultats experimentals confirmen l'eficàcia de les RBFNN, proporcionant estimacions precises i eficients. L'anàlisi comparativa mostra que el model seqüencial AIC + Ridge assoleix un rendiment proper a l'estat de l'art amb un cost computacional molt reduït en comparació amb models més complexos a la literatura científica. A més, la prova d'escalabilitat demostra que el model manté una precisió respectable pel cost computacional que implica, fet que el converteix en una opció sòlida per a conjunts de dades grans que es pot executar còmodament en maquinari de baixa potència.

Paraules clau: Xarxes Neuronals de Funció de Base Radial, agrupament jeràrquic, regularització, eficiència computacional.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context of the project

Radial Basis Function Neural Networks (RBFNNs) emerged as powerful function approximation tools due to their straightforward architecture, allowing easy interpretability and excellent learning capability [1]. Initially proposed with randomly placed neuron centers [2], RBFNNs quickly evolved by adopting clustering-based strategies to position neurons more efficiently. Among these, John Moody and Christian Darken [3] proposed using K-means clustering to find the centers. K-means clustering has become one of the most widely used methods due to its simplicity and computational efficiency. However, K-means presents notable drawbacks, primarily the requirement to predefine the number of neurons—a decision that is often arbitrary and dataset-dependent [4]. To overcome these limitations, our work explores hierarchical clustering as an innovative, data-driven alternative [5]. Hierarchical clustering inherently determines a meaningful number of clusters by analyzing data structures through dendrograms, thus eliminating the need for arbitrary pre-specification. This approach aims to systematically improve neuron placement, enhancing the network's generalization performance and reducing reliance on empirical parameter tuning.

1.2 Objectives

As mentioned in the previous section, the main objective of this project is to develop a novel training method for Radial Basis Function Neural Networks (RBFNNs) that leverages hierarchical clustering for neuron placement and explores different regularisation methods. The goal is to enhance the model's generalization capability and minimize overfitting through the use of advanced regularisation techniques.

To achieve the final result, the project has been divided into several sub-objectives:

- Study the existing RBF neural networks and the common training approaches, identifying weak points and the different associated techniques.
- Investigate the use of hierarchical clustering as a neuron-selection strategy for radial basis function neural networks (RBFNNs), assessing its consistency, complexity and ability to determine the appropriate number of basis functions.
- Study and compare the effects of regularisation techniques within the RBFNN framework, noting the differences between them. Furthermore, compare the analytical solutions with the cross-validation approach for parameter optimization.
- Validate the training approach presented in this work using both synthetic bench-

marks and real-world datasets. Measure predictive accuracy, time complexity and computational scalability.

- Draw conclusions from the comparison and the obtained metrics and summarize the key findings. In particular, determine which model obtains the best evaluation metrics, and finally identify potential extensions.

1.3 Structure of the Document

- The document is organized in different sections.
- Section 1 serves as an introduction, presenting the motivation behind the study, its principal objectives, and the structure of the document.
- Section 2, we review linear models for regression, different model-selection criteria, and regularisation techniques such as LASSO, Ridge, and AIC.
- Section 3 outlines the foundations of Radial Basis Function Neural Networks, including radial functions, network architecture, and the conventional training strategy for RBFNNs.
- Section 4 then introduces Hierarchical Clustering, detailing linkage methods, the clustering algorithm and its complexity, and comparing it against traditional neuron-selection strategies.
- Section 5, we describe our proposed RBFNN training procedure: using hierarchical clustering for centroid placement, optimizing the RBF shape parameter and number of neurons, and applying regularisation strategies.
- Section 6 presents the experimental work: Orr's example, synthetic experiments comparing clustering and regularisation methods, and a realistic application for the method.
- Section 7, we draw conclusions, summarize our contributions, and outline avenues for future work.
- Section A covers Project Management aspects.

2 Background

This section is inspired by Bishop [4, Chapter 3]. Readers are encouraged to consult the reference for further information. The main objective of this section is to present linear models for regression and the important concepts related to them that are relevant to this project.

Linear models form a fundamental class of methods for solving regression problems. The key characteristic of these models is that the output is expressed as a linear combination of fixed basis functions of the input variables. The simplest example of this framework is when the basis functions are the input variables themselves, along with an additional bias term, leading to models that are linear functions of the inputs. By applying non-linear transformations to the inputs via the basis functions, we can model more complex dependencies while still retaining linearity with respect to the parameters $\boldsymbol{w}$. Given an input vector $\mathbf{x}$, the model predicts the corresponding target value $y(\mathbf{x}, \boldsymbol{w})$ through

$$y(\mathbf{x}, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}), \tag{2.1}$$

where $\boldsymbol{w}$ is the vector of weights, and $\boldsymbol{\phi}(\mathbf{x})$ is a vector containing the M basis functions $\phi_j(\mathbf{x})$, which can be, for example, Gaussian or sigmoidal basis functions.

The goal in regression is to determine the parameters $\boldsymbol{w}$ such that the model approximates the relationship between inputs and targets as accurately as possible. This is typically achieved by minimizing an error function, most commonly the sum-of-squares error given by

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y(\mathbf{x}_n, \boldsymbol{w}) - t_n\right)^2, \tag{2.2}$$

where $\{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$ is the training dataset. This error function is proportional to the sum-of-squares error (SSE),

$$\mathrm{SSE} = \sum_{n=1}^{N} \Big(y(\mathbf{x}_n, \mathbf{w}) - t_n\Big)^2, \tag{2.3}$$

and the mean squared error (MSE) is defined as

$$\mathrm{MSE} = \frac{1}{N}\,\mathrm{SSE} = \frac{1}{N} \sum_{n=1}^{N} \Big(y(\mathbf{x}_n, \mathbf{w}) - t_n\Big)^2. \tag{2.4}$$

Linear models for regression offer several advantages, including computational efficiency and well-understood properties. They also form the foundation for more sophisticated models, such as those incorporating regularisation techniques to mitigate overfitting, or probabilistic interpretations that allow for Bayesian linear regression [4].

2.1 Model Selection Criteria

Model selection criteria are used to estimate how well a trained model will generalise to new, unseen data, which is the goal of modelling. These criteria usually rely on measures of prediction error and aim to identify the model that achieves the best balance between fit and complexity. In this section, we review four methods that are later used to select the best model: the Validation Set Approach, Cross-Validation, Generalised Cross-Validation (GCV) and Grid Search.

2.1.1 The Validation Set Approach

The validation set approach is one of the most straightforward strategies for model selection. It consists of randomly splitting the dataset into three disjoint subsets: a *training set*, which is used to fit the model, a *validation set*, which is used to evaluate its predictive performance, and a *test set*, which is used to assess the model's effectiveness. In particular, the test set is employed in each model selection criterion to measure generalization ability. In other sections, the training and validation sets together will be referred to as the learning set. The model is trained using the learning set, and its predictions are then compared with the actual outputs in the validation set.

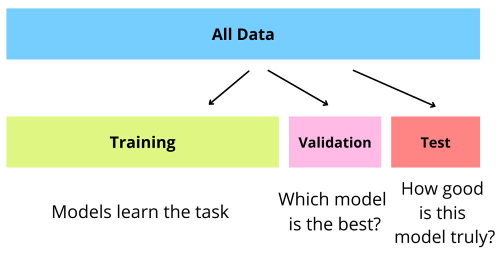


Figure 1: Schematic representation of the validation set approach [6].

The main advantages of this approach are its simplicity and speed. However, this method can be subject to high variance, as the evaluation depends heavily on a single training–validation split. For this reason, more robust alternatives, such as cross-validation, are often preferred.

2.1.2 Cross-Validation

Cross-validation [7] is a technique used to evaluate the predictive performance of a model. The main idea is similar to the validation set approach, but rather than taking into account just one partition of the dataset, cross-validation creates k random partitions of the learning set, as shown in Figure 2.

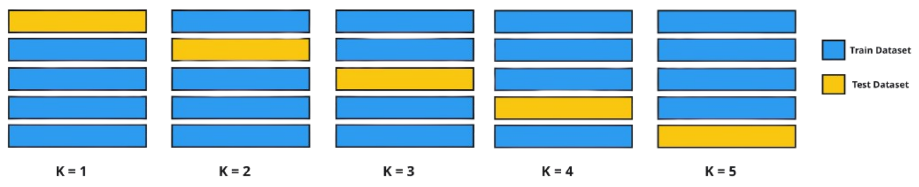


Figure 2: k-fold cross-validation example for $k = 5$ [8].

The model is trained k times, with $k - 1$ subsets used for training each time and the remaining fold used for validation. This avoids any bias that might be introduced by relying on a particular division. The final performance estimate is obtained by averaging the validation errors across all folds:

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE}_i. \tag{2.5}$$

A common variant is Leave-One-Out Cross-Validation [4] (LOOCV), which is a special case of k-fold cross-validation where $k = N$, where N is the number of samples. The primary issue is the high computational cost required, since it trains N models.

2.1.3 Generalised Cross-Validation

As Orr explains in [1, Section 5], the beauty of Leave-One-Out (LOO) for linear models is that it can be calculated analytically by the formula

$$\hat{\sigma}^2_{\mathrm{LOO}} = \frac{\hat{\mathbf{y}}^\top \mathbf{P}\Big(\mathrm{diag}\,(\mathbf{P})\Big)^{-2}\mathbf{P}\hat{\mathbf{y}}}{N}. \tag{2.6}$$

where $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_p]^\top$ is the vector of training set outputs, N is the number of patterns (input-output pairs) and $\mathbf{P}$ is the *projection matrix* (defined later in Eq. 2.8).

However, by replacing the diag($\mathbf{P}$) in the LOO formula with the average diagonal element times the identity matrix,(trace($\mathbf{P}$)$/N$)$\mathbf{I}_p$. The resulting formula becomes numerically more stable. This modification is precisely the Generalised Cross-Validation (GCV):

$$\hat{\sigma}^2_{\mathrm{GCV}} = \frac{p\hat{\mathbf{y}}^\top \mathbf{P}^2 \hat{\mathbf{y}}}{(\mathrm{trace}(\mathbf{P}))^2}, \tag{2.7}$$

The projection matrix equation[1] is

$$\mathbf{P} = \mathbf{I}_N - \mathbf{H}\mathbf{A}^{-1}\mathbf{H}^\top. \tag{2.8}$$

The projection matrix projects the data into the subspace to the model's span. The importance of this square matrix relies on what Orr says in [1, Section 4.2]: "The least squares principle implies that the optimal model is the one with the minimum distance from $\hat{y}$, i.e., the projection of $\hat{y}$ onto the M-dimensional hyperplane." To summarize, this means that solving the least-squares problem is geometrically equivalent to projecting the target vector $\hat{y}$ onto the hyper-plane generated by the M basis functions.

2.1.4 Grid Search

Grid search [9] is a technique used to find the optimal hyperparameters for a machine learning model. It involves an exhaustive search through a subset of previously defined hyperparameters. Therefore, it is important to carefully select the hyperparameters to be tuned, since tuning too many can significantly increase training time without proportionate improvements in performance. The grid search creates all possible combinations of selected hyperparameters and evaluates their performance using a model metric. One example of such a metric is the Normalized Mean Squared Error (NMSE), which is explained in Section 6.2.1.

2.2 Regularisation

Next, we describe regularisation from the perspective of the bias-variance trade-off. This trade-off expresses the relationship between a model's complexity, accuracy and its ability to generalise predictions to unseen data.

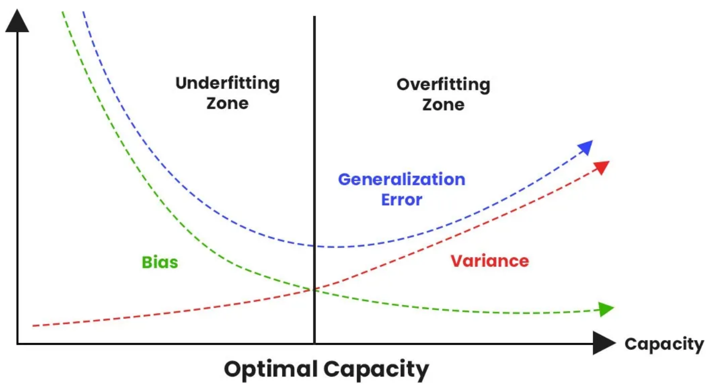


Figure 3: Bias-Variance tradeoff graph [10].

[1]Refer to Section 3.2.3 for the definition on the matrices $\mathbf{H}$ and $\mathbf{A^{-1}}$.

Increasing a model's complexity makes it more flexible, enabling it to fit the training data more closely. This often reduces the training error (i.e. lowers bias). However, if the model is overly complex, it may "memorize" the training examples rather than learning patterns that can be generalized. In that case, its performance on new (unseen) data deteriorates and the variance increases. This phenomenon, whereby a model fits the training set extremely well but fails to generalise, is known as overfitting (see Figure 3).

Regularisation helps to avoid overfitting by preventing the model from becoming overly complex during training. In conclusion, its main objective is to strike the right balance between bias and variance so that the model can generalize well on new data and, in turn, produce accurate results.

In this project, we investigate three main methods of regularisation: LASSO, Ridge and AIC.

2.2.1 LASSO Regularisation

Least Absolute Shrinkage and Selection Operator [11] (LASSO or L1) is a regularisation method that adds a penalty term to the loss function, as shown in Equation 2.9. LASSO works extremely well when we have a large number of features compared to the number of observations, because it shrinks the less important features' coefficients to zero. In other words, it eliminates certain predictors.

$$LASSO(\mathbf{w}) = \sum_{i=1}^{N}\Big(t_i - y(\mathbf{x}_i)\Big)^2 \;+\; \lambda \sum_{j=1}^{M} |w_j|, \tag{2.9}$$

2.2.2 Ridge Regression

Ridge Regression (Tikhonov or L2 regularisation) is a regularisation method that adds a penalty as model complexity increases.

$$Ridge(\mathbf{w}) = \sum_{i=1}^{N}\Big(t_i - y(\mathbf{x}_i)\Big)^2 \;+\; \lambda \sum_{j=1}^{M} w_j^2, \tag{2.10}$$

In general, L2 is appropriate when you want to "smooth" the weights, in other words, it ensures that the weights are small but not zero, unlike LASSO. Reducing the weight is key to avoiding, or at least mitigating, multicollinearity[2].

[2]Multicollinearity [12] occurs when two or more independent variables in a model are linearly related to each other, making it difficult to determine the effect of each variable on the result.

2.2.3 AIC: Akaike Information Criterion

The Akaike Information Criterion [13] is a model selection rule, or criterion, that is intended to estimate the relative quality of statistical models. The formula is

$$\mathrm{AIC} = 2k - 2\ln(\hat{L}), \tag{2.11}$$

where k is the number of estimated parameters and $\hat{L}$ is the maximized likelihood [14] of the model. The combination of both in one equation achieves the balance between complexity and model fit.

In the case of linear regression, under the standard assumption of Gaussianity of model errors, Equation 2.11 boils down to Equation 2.12:

$$\mathrm{AIC} = 2k + N\ln\left(\frac{\mathrm{SSE}}{N}\right), \tag{2.12}$$

where N is the sample size and SSE [15] is the sum of squared error and k is the number of estimated parameters.

One key aspect must be mentioned. The value of AIC is relative to other models and does not indicate the absolute quality of a model. Therefore, if all the models being considered are poor fits, the AIC will not highlight this.

3 Foundations of RBF Neural Networks

This section presents the theoretical basis of RBFNNs, covering their architecture, the functions of radial basis functions, the RBF shape parameter and regularisation.

Radial Basis Function Neural Networks emerged from the realm of numerical interpolation rather than from traditional neural-network research. The effectiveness of RBFs for interpolation was systematically demonstrated in a highly influential survey by Franke [16]. Micchelli's landmark paper provided a comprehensive theoretical justification, proving that for several key RBF types (including multiquadrics and inverse multiquadrics), the interpolation matrix is guaranteed to be non-singular for distinct data points, thus ensuring a unique solution exists [17].

The crucial leap from a pure interpolation method to a neural network paradigm occurred in the late 1980s. In 1988, David Broomhead and David Lowe were the first to formally interpret RBF interpolation as the architecture of a three-layer, feed-forward neural network [2]. In their formulation, the RBF centers of the interpolation method become the prototype vectors of the hidden layer neurons, and the interpolation weights become the weights connecting the hidden layer to the linear output neuron. This reinterpretation was revolutionary, framing a well-understood mathematical technique in the language of neural networks. Almost concurrently, John Moody and Christian Darken proposed a more practical, two-stage learning scheme where the number of centers is independent of the size of the training set, using K-means clustering to find the centers and a pseudo-inverse method to solve for the weights [3]. The final piece of the modern RBF network, the formal inclusion of regularisation theory from a statistical learning perspective (connecting it to Tikhonov regularisation), was articulated by Poggio and Girosi in 1990, providing a framework for managing the bias-variance trade-off [18].

In summary, the genesis of RBF networks is not in neuroscience but in multivariable interpolation, a classical problem in numerical analysis. The core problem can be stated as follows:

Given a set of N distinct data points $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ and a corresponding set of target values $\{y_i \in \mathbb{R}\}_{i=1}^N$, find a function $F : \mathbb{R}^d \to \mathbb{R}$ that satisfies the interpolation condition:

$$F(\mathbf{x}_i) = y_i, \quad \forall i = 1, \ldots, N$$

The RBF approach proposes a solution for $F(\mathbf{x})$ as a linear combination of radially symmetric basis functions, where each function is centered on one of the data points $\mathbf{x}_i$:

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(||\mathbf{x} - \mathbf{x}_i||)$$

where $||\cdot||$ is an Euclidean norm, w_i are the weights to be determined, and $\phi : [0, \infty) \to \mathbb{R}$ is the chosen radial basis function.

By enforcing the interpolation conditions, we obtain a system of N linear equations with N unknown weights:

$$\begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

This can be written in matrix form as $\mathbf{\Phi w} = \mathbf{y}$, where the elements of the interpolation matrix $\mathbf{\Phi}$ are given by $\Phi_{ij} = \phi(||\mathbf{x}_i - \mathbf{x}_j||)$. A unique solution for the weights $\mathbf{w}$ exists if and only if the interpolation matrix $\mathbf{\Phi}$ is non-singular (invertible).

3.1 The Role of the Basis Functions

As explained in the previous section, the choice of ϕ must guarantee that $\mathbf{\Phi}$ is always non-singular for any set of distinct data points. Charles A. Micchelli in his landmark 1986 paper, "Interpolation of scattered data: distance matrices and conditionally positive definite functions." gives the solution to this question.

Micchelli's Theorem provides the conditions for this guarantee. A key result from his work states that for distinct points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, the interpolation matrix $\mathbf{\Phi}$ is non-singular if the basis function $\phi(r)$ is a conditionally positive definite function of a certain order. Many common RBFs satisfy these conditions, for example: inverse quadratic, inverse multiquadric, and the most widely used, the Gaussian function, which is the one we will use. The equation is

$$h(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - c\|^2}{2r^2}\right), \tag{3.1}$$

where r is the radius or shape parameter and c is the neuron center. This function is infinitely differentiable and positive definite. The interpolation matrix is always non-singular. The Gaussian function is illustrated in Figure 5.

3.2 Network Architecture

Radial functions are a class of functions that could be used in any sort of model (linear and nonlinear) and any sort of network (single-layer or multi-layer). However, since David Broomhead and David Lowe [2], radial basis function networks have been associated with radial functions in a single-layer network, such as shown in Figure 4.

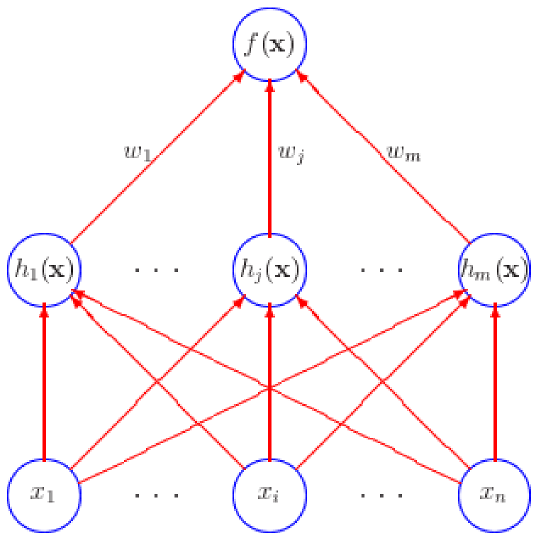


Figure 4: Radial Basis Function Neural Network Architecture [1].

An RBF network is traditionally divided into three layers.

- The **input layer** propagates the input vector x to the hidden layer.
- The **hidden layer** comprises M neurons. The activation of the j-th neuron is computed using an RBF, ϕ_j, based on the distance between the input $\mathbf{x}$ and the neuron's center c_j.
- The outputs of the hidden layer are linearly combined with the weights $\{w_j\}_{j=1}^{m}$ into the network output $f(\mathbf{x})$ called the **output layer**.

This structure transforms the problem from solving a large interpolation system into a two-stage learning problem:

1. **Center and Spread Selection (Unsupervised):** Determine the locations $\mathbf{c}_j$ and spreads r (or shape parameters) for the M hidden neurons. This is often achieved using a clustering algorithm like K-Means on the input data. The centers of the found clusters become the RBF centers.
2. **Weight Calculation (Supervised):** With the basis functions fixed, the hidden

layer outputs for the entire training set are calculated. The problem then reduces to finding the weights **w** that best map these hidden activations to the target outputs. This is a linear problem that can be solved efficiently using methods like the pseudoinverse.

3.2.1 RBF Center Selection

Center selection is a crucial step in training Radial Basis Function Neural Networks (RBFNNs), as it significantly affects the model's ability to make accurate predictions, its ability to generalise, and the computational cost. This step determines the number and the location of the hidden neurons.

The most widely used method for this purpose is K-means [19], which partitions the dataset into K disjoint clusters by minimizing the intra-cluster variance, as detailed in Section 4.3. The resulting centroids are typically selected as the neurons of the model. However, a major limitation of this approach is the requirement to specify the number of clusters K in advance. Choosing a suboptimal value of K can severely impact the performance of the network: if K is too small, the model underfits due to insufficient flexibility; if K is too large, it risks overfitting.

In this work, we adopt hierarchical clustering as an alternative strategy for centroid selection. Unlike k-means, hierarchical clustering constructs a complete dendrogram of nested partitions, allowing the number of clusters to be selected dynamically. The specific approach for centroid selection is explained in Section 5.1.

3.2.2 RBF Spread Selection

Another fundamental variable in Radial Basis Function Neural Networks is the shape parameter, which is often referred to as the smoothing, width or radius parameter.

The choice of the shape parameter plays a critical role in model performance. If the parameter is too large, the radial basis functions become overly smooth, causing the model to behave like a global approximator and potentially lose important local details in the data. On the other hand, if r is too small, each neuron responds only to a narrow region of the input space.

Figure 5 illustrates how the response of a Gaussian RBF monotonically decreases or increases depending on the distance from the center. As Orr [1, Section 3.1] remarks, Gaussian-like RBFs are local, in other words, they give a significant response only in a neighborhood near the center.

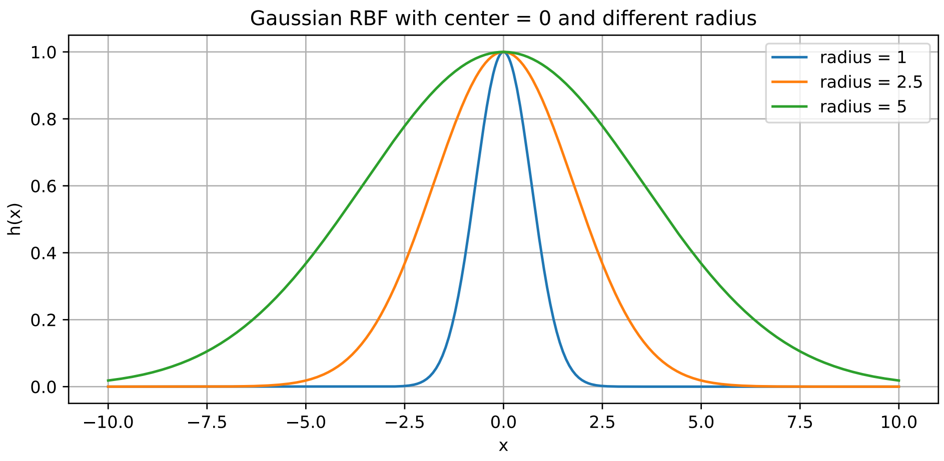


Figure 5: Gaussian RBF response for $radius = [1, 2.5, 5]$ and $center = 0$.

The shape parameter can be assigned globally using a single value for all neurons, or locally, by estimating a separate value for each center. Although using a single global value is simpler and still guarantees universal approximation capability.

Several methods have been proposed in the literature for estimating the shape parameter:

- **Distance-based heuristic [20]:** It defines r as:

$$r = \frac{d_{\max}}{\sqrt{2M}}, \tag{3.2}$$

  where $d_{\max}$ is the maximum Euclidean distance between any two centres and M is the number of centres (hidden neurons). This rule scales r with the overall data span yet shrinks it as $\sqrt{M}$, yielding a roughly constant overlap between neighbouring basis functions.

- **Silverman's rule of thumb [21]:** An estimation method adapted from kernel density estimation. It defines r as:

$$r = \left(\frac{4}{N(d+2)}\right)^{\frac{1}{d+4}} \cdot s,$$

  where N is the number of samples, d is the dimensionality, and s is the standard deviation of the data. This rule tends to perform well when the data is close to normally distributed.

In this study, we examine various approaches to estimating r and analyse their impact on the model's final performance. Generally, these methods do not calculate the optimal shape parameter, so we complement them with a grid search strategy.

Furthermore, each centre c_j can have its own width r_j, which allows for large widths in sparse regions and small ones in dense clusters. While local widths may capture uneven data structures more effectively, they introduce many additional parameters that must be estimated or regularized.

3.2.3 Output Layer Regularisation

Although Micchelli's theorem guarantees a unique solution to the pure interpolation problem, in practice, the interpolation matrix, denoted by $\mathbf{\Phi}$, can be *ill-conditioned* in practice. This occurs when the data points are very close together or when broad basis functions are used. This causes the rows and columns of $\mathbf{\Phi}$ to become nearly linearly dependent. Solving $\mathbf{\Phi}\hat{\mathbf{w}} = \mathbf{y}$ for an ill-conditioned system can lead to a weight vector $\hat{\mathbf{w}}$ with an extremely large norm. The resulting function $F(\mathbf{x})$ will pass through the data points perfectly but may exhibit severe oscillations between them, leading to very poor generalization on new data. This is a classic example of overfitting.

For this reason, regularisation is necessary, since it introduces a penalty for model complexity, in order to solve this ill-posed problem by enforcing a "smoother" solution.

How Regularisation is Implemented

In this section, it is explained how Orr implements regularisation using the most common approach, called Tikhonov regularisation or ridge regression. For clarity, Table 1 summarizes the main symbols and their dimensions used throughout this section.

Symbol	**Dimension**	**Meaning**
$\mathbf{H}$ $(= \mathbf{\Phi})$	$N \times M$	Design matrix of RBF
$\hat{\mathbf{w}}$	M	Output weight vector
$\mathbf{A^{-1}}$	$M \times M$	Variance matrix, Equation 3.4
$\mathbf{P}$	$N \times N$	Projection matrix, Equation 2.8
$\mathbf{\Lambda}$	$M \times M$	$\lambda \mathbf{I}_M$
N	scalar	number of training patterns
M	scalar	number of RBF centers (neurons)

Table 1: Notation used. Dimensions assume N training patterns and M RBF centers.

The objective function to be minimized is Equation 2.10. To find the optimal weight vector $\hat{\mathbf{w}}$, Orr [1, Section 4] uses a linear pseudo-inverse solution. The equation is

$$\hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{H}^{\top}\hat{\mathbf{y}}, \tag{3.3}$$

where $\mathbf{A^{-1}}$ is the *variance matrix* and its equation is

$$\mathbf{A}^{-1} = \left(\mathbf{H}^{\top}\mathbf{H} + \mathbf{\Lambda}\right)^{-1}, \tag{3.4}$$

so, we plug Equation 3.4 into Equation 3.5 and obtain the next equation

$$\hat{\mathbf{w}} = \left(\mathbf{H}^{\top}\mathbf{H} + \mathbf{\Lambda}\right)^{-1}\mathbf{H}^{\top}\hat{\mathbf{y}}, \tag{3.5}$$

$\mathbf{\Lambda}$ is a diagonal matrix with the regularisation[3]parameter along its diagonal, $\hat{y}$ is the vector of training set, $\mathbf{H}$ is the *design matrix*,

$$\mathbf{H} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \dots & h_M(x_1) \\ h_1(x_2) & h_2(x_2) & \dots & h_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(x_N) & h_2(x_N) & \dots & h_M(x_N) \end{bmatrix}. \tag{3.6}$$

$\mathbf{H}$ contains all the RBFNN results, where h_i correspond to a neuron in the *hidden layer.*

The matrix $(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})$ is guaranteed to be invertible as long as $\lambda > 0$. The addition of the positive term $\lambda\mathbf{I}$ to the diagonal makes the matrix well-conditioned and stabilizes the matrix inversion, yielding a robust solution that generalizes better to unseen data. Therefore, the choice of λ is critical.

It is important to note that Orr only uses and studies Ridge regression. Therefore, this analytic solution cannot be applied to other regularisation methods such as Lasso.

Optimization for regularisation parameter

Previously, we explained the Equation 3.5 used for Orr in his work. In this equation appears the component $\mathbf{\Lambda}$, which contains the regularisation parameter. This parameter penalizes the weight equation; consequently, some kind of optimization is necessary. Otherwise, we risk applying too much or too little penalty, resulting in underfitting or overfitting of the model, respectively.

The most common optimization method is Cross-Validation, detailed in Section 2.1.2, However, CV is computationally expensive. For this reason, we will explore other analytic alternatives.

To optimise the regularisation parameter, Orr suggests using the *Generalized Cross-Validation*, explained with more detail in Section 2.1.3. The Equation 2.7 can be ma-

[3]Orr is applying ridge regression. If regularisation is not needed, the regularisation parameters are set to zero. In this case, the equation performs ordinary least squares.

nipulated into the Equation 3.7. This ensures that only one occurrence of the variable $\hat{\lambda}$ is on the left-hand side.

$$\hat{\lambda} = \frac{\hat{\mathbf{y}}^\top \mathbf{P}^2 \hat{\mathbf{y}} \operatorname{trace}\left(\mathbf{A}^{-1} - \hat{\lambda}\mathbf{A}^{-2}\right)}{\hat{\mathbf{w}}^\top \mathbf{A}^{-1} \hat{\mathbf{w}} \operatorname{trace}(\mathbf{P})}. \tag{3.7}$$

$\mathbf{P}$ is the projection matrix (defined in Equation 2.8), $\hat{\mathbf{y}}$ is the vector of training set and $\mathbf{A}^{-1}$ is the variance matrix In addition, Orr remarks that this equation is not a solution for $\hat{\lambda}$, it is a re-estimulation formula. The correct usage for the formula is, first, choosing an initial $\hat{\lambda}$, and then using it in the right-hand side for calculate a new $\hat{\lambda}$ value in the left-hand side. This results in a new estimate, and the process can be repeated until convergence is achieved.

To summarize, it could be used two different approaches using Equation 2.7 and Equation 3.7. These two methods both stem from the same equation, but by rewriting it in two different forms, we obtain two distinct approaches.

4 Hierarchical Clustering

Hierarchical clustering is a fundamental unsupervised learning technique that is used extensively for clustering and data analysis to identify intrinsic structures within data. Hierarchical clustering can be divided into two categories [22], as shown in Figure 6:

- **Agglomerative** (bottom-up): starts with each data point forming an individual cluster. In each iteration, the algorithm merges the two most similar clusters based on a distance metric and a linkage criterion. The process ends when all clusters have been combined into one. The agglomerative strategy is more commonly used because of its simplicity and computational efficiency.
- **Divisive** (top-down): starts with one cluster containing all the data points. At each iteration, the cluster is split into smaller ones using a distance criterion. Divisive clustering is used when the initial objective is to identify large, distinct clusters.

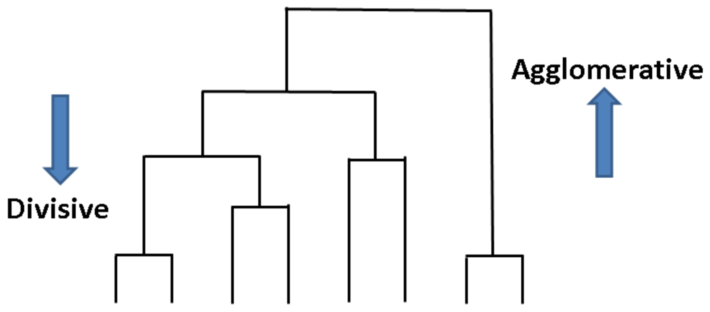


Figure 6: Dendrogram for hierarchical clustering type [23].

Hierarchical clustering offers a visualization tool in the form of a dendrogram, which can be used to assess data relationships and to select clusters based on criteria such as the distance between groups or the desired granularity of the clusters. In this project, hierarchical clustering is employed to make an initial estimate of the neurons that best represent the input data distribution.

4.1 Cluster Linkage

In this section, we will explain the cluster linkage methods that exist for hierarchical clustering. Selecting an appropriate metric for linkage is a crucial part of the clustering process, as different choices can result in clusters of very different shapes, as illustrated in Figure 7.

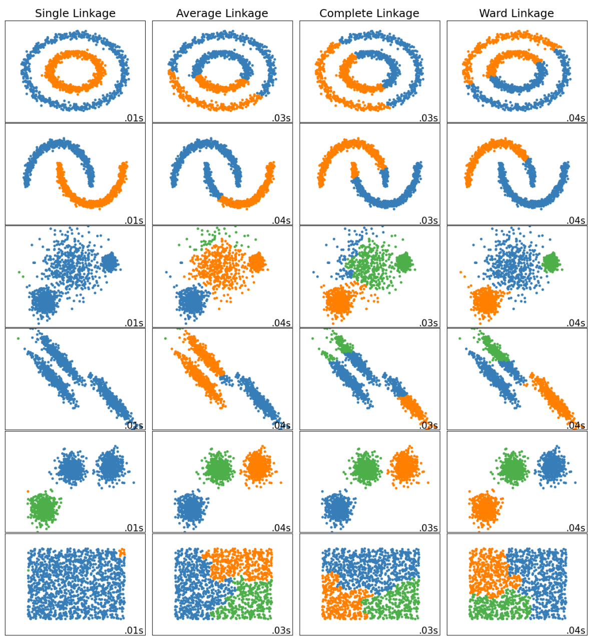


Figure 7: Result comparison between linkage criterion using: Single Linkage, Average Linkage, Complete Linkage and Ward Linkage [24].

As Figure 7 illustrates, all the linkage criteria have their pros and cons. Firstly, single linkage is ideal for joining points along elongated or non-convex shapes. It is also the fastest method, but it struggles with noisy datasets. Secondly, average linkage works slightly better than single linkage with non-convex shapes and noisy datasets, but the improvement is not significant and it is slow. In third place, complete linkage produces very compact clusters, but it performs badly with elongated shapes. Finally, ward linkage is perfect for compact clusters and is highly stable when dealing with noisy data; however, it is the slowest. In summary, different linkage criteria exhibit different behaviors. Therefore, selecting the appropriate criterion is crucial for hierarchical clustering.

Table 2 shows the formulas for linkage criteria [5] mentioned previously. We define A and B as two observation sets with a distance function (metric) d:

Names	**Formula**
Single linkage [25]	$d_{\mathrm{SL}}(A,B) = \min\limits_{a\in A,\, b\in B} d(a,b)$
Complete linkage [26]	$d_{\mathrm{CL}}(A,B) = \max\limits_{a\in A,\, b\in B} d(a,b)$
Average linkage [27] (UPGMA)	$d_{\mathrm{AL}}(A,B) = \dfrac{1}{\lvert A\rvert\,\lvert B\rvert} \sum\limits_{a\in A} \sum\limits_{b\in B} d(a,b)$
Ward linkage [28]	$d_{\mathrm{Ward}}(A,B) = \dfrac{\lvert A\rvert\,\lvert B\rvert}{\lvert A\rvert + \lvert B\rvert} \left\lVert \boldsymbol{\mu}_A - \boldsymbol{\mu}_B \right\rVert^2$

Table 2: Common linkage criteria and their distance formulas. $\boldsymbol{\mu}_A$ and $\boldsymbol{\mu}_B$ are the centroids of A and B.

The formulas provide more clarity on why they behave as observed in Figure 7. Single linkage merges clusters based on their closest pair of points, while complete linkage merges clusters based on their farthest pair of points. Due to their simplicity, they produce poor results with noisy data. Average linkage strikes a balance between single and complete linkage. Finally, ward linkage minimizes the increase in within-cluster variance at each merge.

4.2 The Algorithm and Complexity

The complexity of an algorithm is a key consideration. While several complex algorithms can produce great results, they are usually too computationally expensive. Therefore, it is important to strike the right balance between algorithm complexity and results. Next, we will study the Agglomerative Hierarchical Clustering Algorithm [4]:

Algorithm 1 Agglomerative Hierarchical Clustering

```
1: Input: Data $X_N$.
2: Calculate distance matrix, $D$: $\forall$ pair of indexes $(i, j) \in X_N$ with $1 \leq i < j \leq N$,
   calculate $D[i, j] = \text{dist}(x_i, x_j)$
3: Initialize: $\mathbf{Z} = \emptyset$, $C \leftarrow \{x_1, \dots, x_N\}$ {Each point is a cluster.}
4: while $|C| > 1$ do
5:     Find the pair $(x_i, x_j)$ with the min distance in $D$.
6:     $x_u \leftarrow x_i \cup x_j$ {Merge clusters into a new one.}
7:     Update the $\mathbf{Z}$ with the new cluster, $Z \leftarrow Z \cup x_u$.
8:     $C \leftarrow \Big(C \setminus \{x_i, x_j\}\Big) \cup x_u$ {Update $C$}
9:     Update the distance matrix with the new distances using a linkage criterion.
10: end while
11: return Z
```

As we can observe in Algorithm 1, we first consider all data points as clusters. We then merge them until all single clusters are unified into one. In general, Z is a data structure (for instance, a matrix) with $N - 1$ rows, where each row represents a merge of two different clusters. This is done because it is very simple to create a dendrogram from Z to visualize the clustering.

Now, we will discuss the time complexity. If we take a look at Algorithm 1, we start by computing the distance matrix, which consists of an $N \times N$ matrix containing the distances[5] between all data points. Therefore, the time complexity is $\mathcal{O}(n^2)$.

Next, we will analyse the merge loop. After computing the distance matrix, the algorithm enters a loop that performs $N - 1$ iterations, as exactly $N - 1$ merges are needed to reduce

[4]Divisive Hierarchical Clustering Algorithm is not explained because we do not use it in the experimental work, however, the main idea is similar to the Agglomerative approach.

[5]In general, the Euclidean distance is used.

N singleton clusters to one. In each iteration, the algorithm selects the pair of clusters that are closest according to the chosen linkage criterion, merging them to form a new cluster. This process requires updating the distances between the newly formed cluster and all the remaining clusters to be updated.

Using a naive implementation takes $O(n^2)$ time per iteration to identify the closest pair of clusters, since all pairwise distances must be checked. Additionally, updating the distances to the new cluster also requires scanning up to $\mathcal{O}(n)$ clusters, depending on the linkage criterion.

As a result, the overall complexity of the merge loop becomes $\mathcal{O}(n^2)$ per iteration over N iterations, leading to a total time complexity of $\mathcal{O}(n^3)$. However, optimised implementations[6] use priority queues and efficient data structures to reduce this to $\mathcal{O}(n^2 \log n)$.

4.3 Comparison with Traditional Neuron Selection

Several clustering techniques have been employed in the literature to select neuron centers in Radial Basis Function Neural Networks (RBFNNs). Methods such as BIRCH [30], CURE [31], and DBSCAN provide different approaches to identify data structures and group similar instances. Nevertheless, k-means clustering remains the most commonly used and computationally efficient method.

K-means Clustering

The k-means algorithm [19] partitions a data set $X = \{x_1, x_2, \ldots, x_n\}$ into k disjoint clusters $C_1, C_2, \ldots, C_k$ by minimizing the within-cluster sum of squared distances. Formally, the objective is to find cluster centers $\mu_1, \mu_2, \ldots, \mu_k$ such that the following cost function is minimized:

$$S = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2.$$

The algorithm proceeds iteratively by alternating between two steps:

- **Assignment step**: Assign each point $x \in X$ to the cluster whose centroid is closest.
- **Update step**: Recompute the centroid of each cluster.

These steps are repeated until convergence occurs, typically when the assignment of the clusters no longer changes or the improvement in S falls below a threshold.

[6] Our Python code follows that approach. Refer to the SciPy [29] documentation and Müllner [5] for implementation details.

Comparison with Hierarchical Clustering

Although k-means is simple and efficient, it requires the number of clusters k to be specified in advance. This poses a significant limitation in practice, particularly when the intrinsic structure of the data is unknown. Choosing an incorrect value for k can result in underfitting (too few clusters) or overfitting (too many clusters).

By contrast, hierarchical clustering creates a dendrogram representing nested groupings of the data, without the need for a predefined number of clusters. The number of clusters can be chosen dynamically by analyzing the dendrogram visually or by applying a stopping criterion, eliminating the need to recalculate the entire algorithm. Furthermore, hierarchical clustering provides a richer structural representation of the data with the dendrogram, Figure 6.

In summary, the main issue with K-means is its reliance on a fixed k, limiting its applicability in scenarios where the number of clusters is unknown a priori. In contrast, hierarchical clustering overcomes this limitation and enables a more adaptive neuron selection process.

5 Training of the RBF Neural Network

In this section, we will discuss the design of the Radial Basis Function Neural Network (RBFNN). It has been decided to implement it in Python [32] because of the large number of machine learning libraries available and because the author has prior knowledge of Python. Despite the existence of a few RBFNN implementations, we have decided to create a new one. This is because we need to adapt the implementation to our strategy. In summary, we require flexibility when selecting centroid calculation strategies and regularisation methods in order to be able to compare them, two aspects that previous implementations do not facilitate.

5.1 Hierarchical Clustering for Centroid Selection

The number and placement of neurons in the *hidden layer* are crucial factors in the implementation of an RBFNN. In this study, hierarchical clustering is employed to determine the placement of neurons. Since one of the main objectives is to evaluate different regularisation methods, the model should be slightly overfitted before regularisation is applied.

To this end, we adopt the following heuristic: *select the approximate*[7] *minimum number of neurons, denoted by H, that causes the model to overfit.* The algorithm is the following:

Algorithm 2 Centroid Selection for RBFNN

```
Require: $X_n$ {Input data}, $m$ {minimum number of neurons}, $r$ {Shape Parameter}
Require: $\alpha$ {increment}, $\tau$ {threshold}.
 1: Compute pairwise Euclidean distances of $X$. {Matrix $n \times n$}
 2: Create the dendrogram with Algorithm 1.
 3: while $m < n$ do
 4:    Cut the dendrogram to obtain $m$ clusters.
 5:    Calculate the centroids $C_m$ for each cluster.
 6:    Train the RBFNN with $r$ and $C_m$.
 7:    if $\text{NMSE}_{validation}/\text{NMSE}_{training} > \tau$ then
 8:       return $m$.
 9:    else
10:       $m = m \cdot \alpha$
11:    end if
12: end while
13: return Overfitting not found.
```

To determine if the model is overfitted, we calculate the Normalized Mean Squared Error (NMSE) and divide the testing set values by the training set values to obtain a ratio.

[7]Finding the exact minimum H that overfits would be too expensive and unnecessary.

Then, we define a *threshold*[8] for the ratio that indicates whether the model has been overfitted or not. If overfitting is not detected and the number of neurons m is greater than the number of data points n, we decrease the *threshold* and execute the algorithm again. However, this would only happen if we select a large threshold τ.

The Figure 8 provides a graphical demonstration of the concept of cutting the dendrogram to produce different clusterings with a small group of data. Specifically, the dendrogram was cut three times to produce two, three and four clusters, respectively.

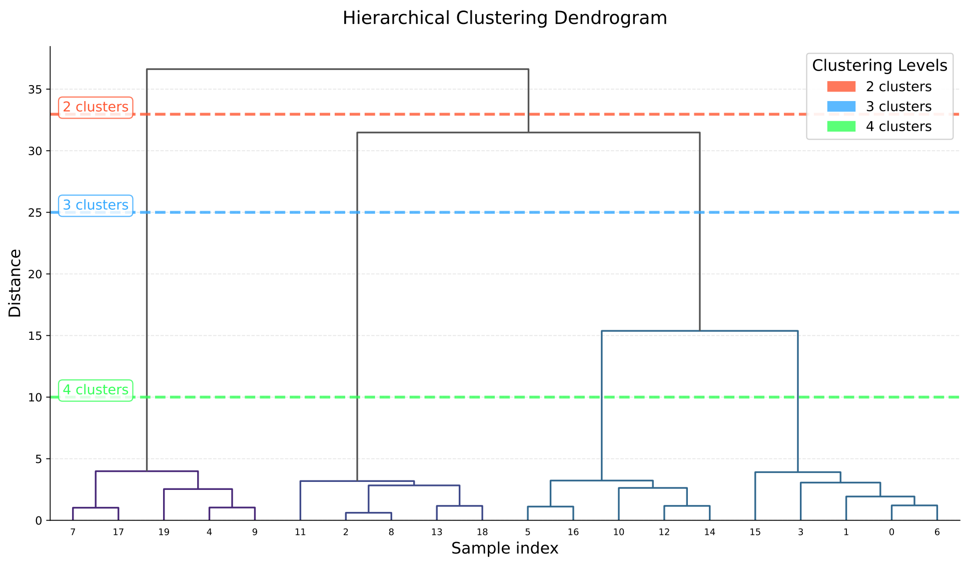


Figure 8: Representation of cutting the dendrogram to obtain 2, 3 and 4 clusters.

Finally, once we have calculated the number of neurons H, it is necessary to place them. For this step, we opt for K-means for the following reasons: K-means clustering is computationally efficient and scales well with data size. Furthermore, we know exactly how many clusters are required.

5.2 Optimization for the RBF Shape Parameter and the number of neurons

In the previous section, we described the algorithm for selecting the number of neurons H based on the proposed heuristics[9].However, we did not explain how the shape parameter was defined to train the model in that algorithm. The shape parameter plays a crucial role in our heuristic, meaning it is impossible to determine the number of neurons without an appropriate width parameter.

Moreover, the shape parameter depends entirely on the dataset used. Therefore, we need

[8]In general, we define a threshold of 1.5, but if overfitting is not detected, we reduce the value.
[9]The approximate minimum number of neurons that causes the model to overfit.

a range of values that are adapted to the dataset. For that reason, we opted to apply the methods in Section 3.2.2 to obtain an r and use it as a pilot value. This pilot value serves as the center of a range of candidate values created by multiplying it by several factors to explore a slightly smaller and larger radius. For example, if the pilot r is 0.5, we define candidates like $0.5 \times 0.8 = 0.4$, and $0.5 \times 1.2 = 0.6$. In general, the pilot value produces acceptable results, but exploring a range of values allows better optimization.

Algorithm 3 details the dual optimization using this grid search over both the shape parameter and the number of neurons.

Algorithm 3 Shape Parameter and Neuron Count Optimization for RBFNN

```
1: Compute the pilot shape parameter $r_0$ using the methods in Section 3.2.2.
2: Define the grid of candidate shape parameters:
       $R = \left\{ r = \alpha \cdot r_0 \;\middle|\; \alpha \in \mathcal{A} \right\}$
   where $\mathcal{A}$ is a set of multipliers.
3: for each $r \in R$ do
4:     Determine the optimal number of neurons $H_r$ by executing Algorithm 2 with radius $r$.
5:     Evaluate $\mathrm{NMSE}_{\mathrm{val}}(r, H_r)$ on the validation set.
6: end for
7: return $(r^*, H^*)$ that minimizes $\mathrm{NMSE}_{\mathrm{val}}(r, H)$
```

The time complexity of the Algorithm 3 depends on two aspects. The first aspect is the number of multipliers. In particular, there are $|\mathcal{A}|$ of multipliers. The other aspect is the Algorithm 2 and the cost of training the model T_{train}, since the algorithm trains a model to compute the metrics and evaluate them. Finally, the grid search cost is

$$\mathcal{O}\left(|\mathcal{A}| \times (T_{\text{train}} + T_{\text{Alg2}})\right), \tag{5.1}$$

5.3 Regularisation methods

The final stage in training a Radial Basis Function Neural Network (RBFNN) is to calculate the weights and apply regularisation to prevent overfitting.

The following is a discussion of the various regularisation methods that have been implemented. In particular, we will describe each algorithm in detail and explain why we opted for these sequential combinations over others.

5.3.1 Basic Strategies

In this section, we will explore Ridge, Lasso and AIC, which are three common and well-researched regularisation strategies. Our decision to focus on these techniques rather than others is based on the potential for sequential combination. This is described in Section 5.3.2.

Ridge Regression

Ridge Regression is the most important method because we evaluated the analytical version presented by Orr. As we mentioned earlier in Section 3.2.3, we will study two different approaches.

The first strategy consists of the GCV loop. We compute the GCV Equation 2.7 with different values of λ with the objective of finding the λ value, which minimizes the GCV error. In particular, a logarithmic search is employed to define the candidates. The candidates are:

$$\mathbf{\Lambda} = \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}, 10^{1}\}.$$

The algorithm is:

Algorithm 4 Ridge Regression with GCV loop

```
1: Select $\mathbf{\Lambda} = \{\lambda_1, \ldots, \lambda_K\}$ to test.
2: for each $\lambda_i$ in $\mathbf{\Lambda}$ do
3:     Compute the GCV error for $\lambda_i$.
4: end for
5: return $\lambda^*$ with the $\min(\hat{\sigma}^2_{\text{GCV}})$.
```

During the execution of the algorithm, the GCV error is illustrated graphically as shown in Figure 9. This allows us to visualize the results and quickly detect any anomaly.

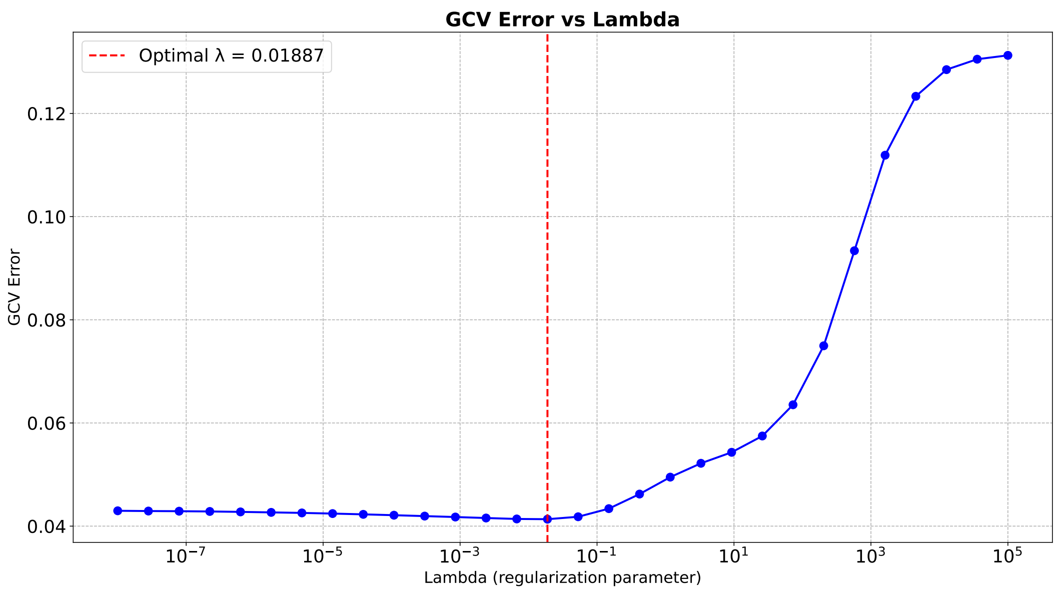


Figure 9: Plot of the GCV error for different values $\mathbf{\Lambda} = \{10^{-8}, \ldots, 10^{5}\}$ using `make_regression` [33] with $n_{\text{samples}} = 1000$, $n_{\text{features}} = 4$, and Gaussian noise $= 1.2$. The range of λ values was increased to enhance the plot's resolution.

The second strategy uses the re-estimation Equation 3.7. The algorithm is:

Algorithm 5 Ridge Regression with re-estimation formula

```
1: Select $\mathbf{\Lambda} = \{\lambda_1, \ldots, \lambda_K\}$ to test.
2: for each $\lambda_i$ in $\mathbf{\Lambda}$ do
3:     Compute the re-estimation formula for $\lambda_i$.
4: end for
5: Calculate the GCV error for the converged values of $\lambda$.
6: return $\lambda^*$ with the $\min(\hat{\sigma}^2_{\text{GCV}})$
```

The final Ridge regression method is the function *RidgeCV* [34] from the Python library *scikit-learn* [35]. This method selects the optimal value of λ using Cross-Validation with a logarithmic search. As we implement two analytic methods, it is important to ensure that the new implementation works properly. For this reason, we have decided to include *RidgeCV* to enable us to compare our methods with the most common alternatives for ridge regression.

LASSO Regularisation

As we explained in Section 2.2.1, the key benefit of LASSO is its ability to reduce the coefficients of less important features to zero. This enables us to reduce the number of neurons in the model.

LASSO does not have an analytic solution as Ridge does. Therefore, we use the LASSO implementation in *scikit-learn* [35]. In particular, we employ the function *LassoCV* [36], which automatically selects the optimal value of λ using cross-validation.

AIC: Akaike Information Criterion

As AIC is a criterion for comparing models, the best practice is to combine it with other subset selection algorithms to enable comparison between several models. In this project, we apply the *Backward Stepwise Selection* algorithm. The combined algorithm is the following:

Algorithm 6 Backward Elimination Based on AIC

```
1: Let $F_n$ be the full model containing all $n$ predictors.
2: for $i = n, n-1, \dots, 1$ do
3:     Generate all candidate models $F_{i-1}$ by removing one predictor from $F_i$.
4:     for each candidate $F_{i-1}$ do
5:         Fit the model to the training data and compute its AIC.
6:     end for
7:     Select $F_{i-1}$ as the model with the lowest AIC among the candidates.
8: end for
9: Choose the final model as the one with minimal AIC among $\{F_0, F_1, \dots, F_n\}$.
```

The other strategy used to reduce the number of neurons in the model, apart from Lasso, is AIC. However, there is an important and obvious difference in the computational cost. LASSO reduces the number of neurons by adjusting a global optimization problem. In contrast, AIC with backward stepwise selection is a greedy algorithm, which is generally more expensive. Furthermore, a global optimal solution is not guaranteed due to the possibility of local minima.

The local minima problem [37] occurs when the function finds a minimum that is lower than all nearby points but not the lowest possible value across the entire search space. This is known as the global minimum. This causes the algorithm to 'get stuck' in a suboptimal solution, as it has no mechanism to escape and explore better alternatives. A graphical representation is provided in Figure 10.

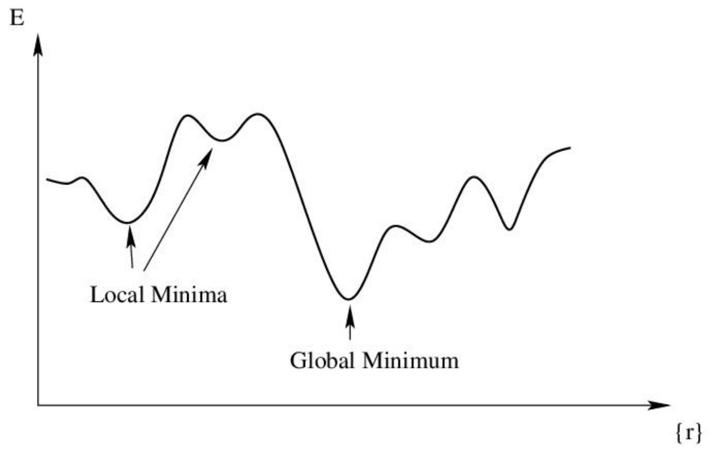


Figure 10: Graphical representation of the local minima problem [38].

5.3.2 Sequential Strategies

The main idea behind sequential strategies is to exploit the strengths of two different regularisation methods by executing them sequentially. Specifically, the first method aims to reduce the number of neurons. The second method focuses on shrinking the weights.

Some methods in the literature also combine two regularisation techniques, but with a different approach. Elastic Net [39] is the most well-known of these. Elastic Net is a regularisation technique that linearly combines the LASSO and Ridge methods, both detailed in Sections 2.2.1 and 2.2.2, respectively. The formula is

$$ElasticNet(\mathbf{w}) \;=\; \sum_{i=1}^{N}\Big(t_i - y(\mathbf{x}_i)\Big)^2 \;+\; \lambda_1 \sum_{j=1}^{M} |w_j| \;+\; \lambda_2 \sum_{j=1}^{M} w_j^2. \tag{5.2}$$

The Elastic Net approach has several advantages. In first place, addresses multicollinearity[10] by either excluding or retaining groups of highly correlated predictors. Furthermore, the combination of the L_1 and L_2 parameters strikes a balance between bias and variance. However, the main issue with Elastic Net is that it is more difficult and expensive to tune due to its two parameters. For this reason, we explore the sequential approach, as we also aim to tune two parameters, but not simultaneously, as Elastic Net does.

In summary, our approach will enable us to adjust two different parameters independently, improving our computational implementation. Even though tuning the two parameters simultaneously, as Elastic Net does, has the advantage of finding the global optimum.

[10]Multicollinearity [40] occurs when two or more independent variables in a model are closely related to each other, making it difficult to determine the effect of each variable on the result.

Another important aspect to note is that not all possible sequential combinations of regularisation methods are meaningful. For instance, when two consecutive stages both aim to prune neurons, such as running Lasso followed by AIC, the second stage becomes largely redundant. Lasso has already eliminated neurons with weights that can be reduced to zero, so AIC can only eliminate a few more neurons while inevitably increasing the computational cost.

Similarly, using Ridge after Lasso or AIC also holds limited value. In fact, the naive version of elastic net employed this approach. However, this type of estimation results in double the amount of shrinkage, leading to increased bias and inaccurate predictions.

Sequential Lasso + Ridge

In this Section, we explain the Sequential Lasso + Ridge algorithm. Three different input parameters are needed. In first place, the number of Neurons H and the shape parameter 3 are obtained with Algorithm 3. Finally, the Ridge method is chosen between the three strategies: 'GCV loop', 're-estimation formula' or 'RidgeCV'.

Algorithm 7 Sequential Lasso + Ridge

```
1: **Input:** number of neurons $H$, shape parameter $r$, and a ridge method $R$.
2: Apply LassoCV [36] to estimate the weights according to Equation **??**.
3: Eliminate all neurons whose weights have been shrunk to zero.
4: Apply the ridge method $R$ using the reduced set of neurons.
5: **return** The model metrics.
```

First, we apply Lasso to reduce the total number of selected neurons with the intention of overfitting the model. Then, we apply Ridge to the reduced number of neurons.

Sequential AIC + Ridge

In this Section, we explain the Sequential AIC + Ridge algorithm. The input parameters are the same from Algorithm 7.

Algorithm 8 Sequential AIC + Ridge

```
1: **Input:** number of neurons $H$, shape parameter $r$ and ridge method $R$.
2: Use Algorithm 6 to determine the number of reduced neurons $h$.
3: Execute the Ridge method $R$ with the number neurons equal to $h$.
4: **return** The model metrics.
```

As in the previous section, we first apply a strategy to reduce the number of neurons, and then apply the Ridge method.

5.4 Proposed Training Workflow

In this section, we illustrate the proposed training workflow.

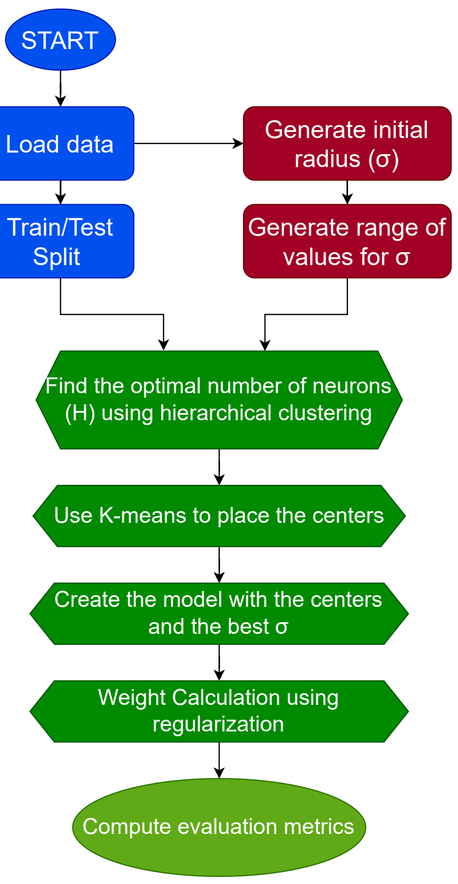

START
Load data
Generate initial radius (σ)
Train/Test Split
Generate range of values for σ
Find the optimal number of neurons (H) using hierarchical clustering
Use K-means to place the centers
Create the model with the centers and the best σ
Weight Calculation using regularization
Compute evaluation metrics

Figure 11: Training Workflow for the RBFNN.

State	**Detailed in**
Load data	Section 6
Train/Test Split	Section 2.1.1
Generate initial radius (r)	Section 3.2.2
Generate range of values for r	Section 5.2
Find optimal H using clustering	Section 5.1
Use K-means to place centers	Section 4.3
Create model with centers and best r	Section 5.2
Weight Calculation using regularisation	Section 5.3
Compute evaluation metrics	Section 6.2.1

Table 3: Mapping of training process blocks to report sections.

6 Experimental Work

The experimental work is divided into three parts. The first will verify that our implementation works correctly. To achieve this, we first simulate the problem posed by Orr in his paper to ensure that we obtain comparable results, since exact reproduction is impossible due to the additive Gaussian noise. In second place, we will attempt to predict a one-variable function to observe the model's behaviour in each instance with synthetic data. Finally, we will use real data to evaluate the performance of Radial Basis Function Neural Network (RBFNN) in a real application.

6.1 Orr's Example

In this section, we reproduce Orr's example [1, Section 6.5]. We will follow all the steps indicated by the author and compare our results with his. We will now consider the proposed example.

Orr chooses for the training set $N = 50$ randomly sampled patterns from the sine wave

$$y = \sin(12x) \tag{6.1}$$

with the points between $x = 0$ and $x = 1$ with Gaussian noise of standard deviation $\sigma = 0.1$ added. The model used is a radial basis function network with $M = 50$ Gaussian function of width $r = 0.05$ whose position coincide with the training set input points. Each fit uses a standard ridge regression with four different values for the regularisation parameter $\lambda \in \{10^{-10}, 10^{-5}, 1, 10^5\}$. Figure 12 shows the fit for each value of λ.

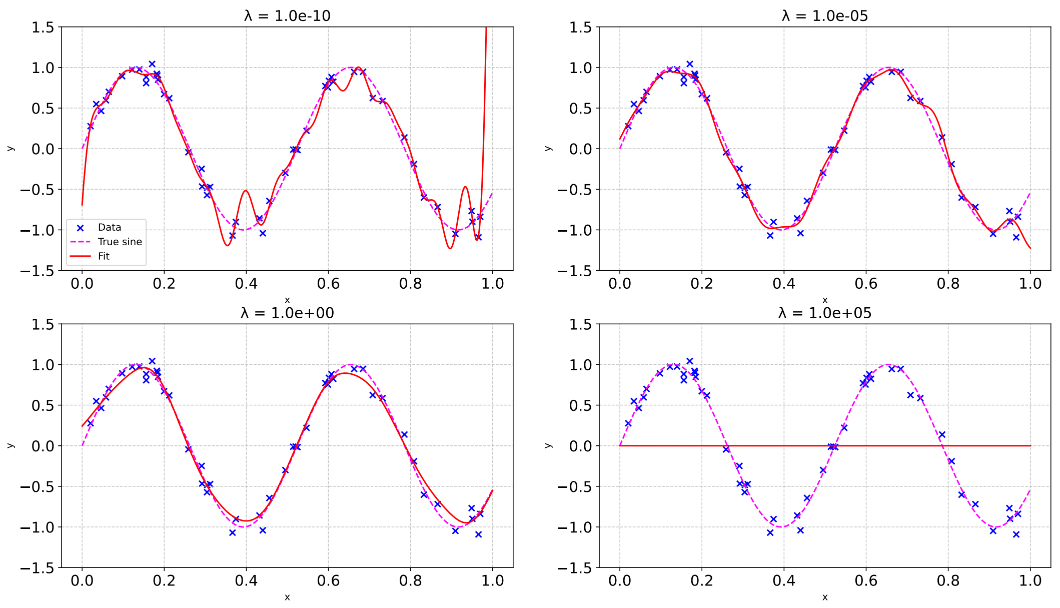


Figure 12: Four different RBF fits (solid red curves) to data (blue crosses) sampled from a sine wave (dashed magenta curve).

Orr's next step is to study how RMSE (root-mean-squared-error) behaves according to the value of lambda. Specifically, he calculates the RMSE using an array of 250 noiseless samples of the target between $x = 0$ and $x = 1$. As can be seen in Figure 13, the best value is $\lambda \approx 0.1$.

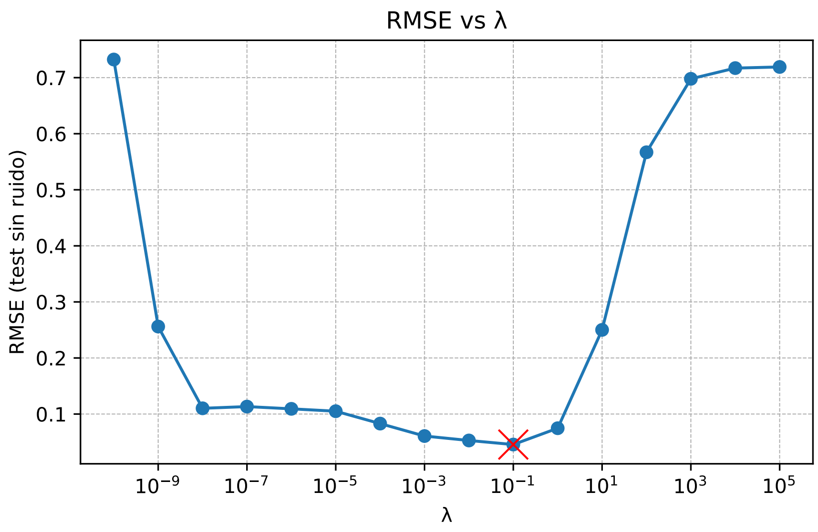


Figure 13: RMSE as function of λ. The optimal value $\lambda = 0.1$ is shown with a star.

The problem, as Orr suggests, is that in real applications, the target is unknown and therefore we do not have access to the RMSE. For that reason, we have to use some kind of model selection criteria to find parameters such as λ. The author suggests and explores two methods. The first is to evaluate how the GCV, Section 2.1, varies over a range of λ values. As we can see, Figure 14 suggests $\lambda \approx 0.1$ as the optimal value.

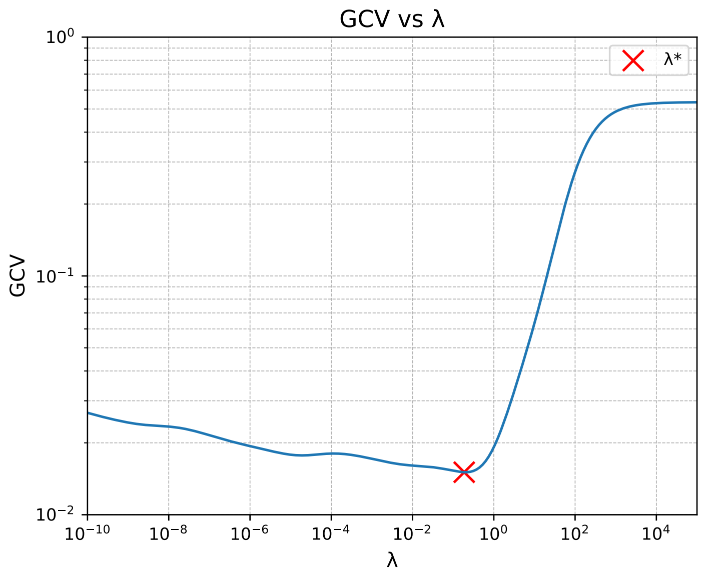


Figure 14: GCV as function of λ with the optima value $\lambda \approx 0.1$ shown with a star.

Finally, we will use the re-estimation formula based on GCV, Section 3.2.3. In Table 4, we compare the values obtained by Orr and ours.

	First iteration		**Second iteration**	
	Initial λ	Optimal λ	Initial λ	Optimal λ
Orr	0.01	0.1	10^{-5}	$2.1 \cdot 10^{-4}$
Ours	**0.01**	**0.146**	$\mathbf{10^{-5}}$	$\mathbf{2 \cdot 10^{-5}}$

Table 4: Comparison of λ values using re-estimation formula.

As reported in Table 4, we obtain similar results. In order to understand the results, we should also look at Figure 14. What Orr tries to show is that this formula is strongly conditioned by the initial λ, since in certain cases it can reach a local minimum, which would lead to a sub-optimal solution. In summary, it is important to try different initial lambdas to avoid the problem of local minima, explained in Section 5.3.1.

Now, we are going to discuss why there are differences between them. First, we must remember that what Orr is showing is an example and not an experiment. Only one iteration of the entire method is performed. This, coupled with the fact that we are dealing with data containing Gaussian noise, makes it impossible to draw firm conclusions if we do not use averages across different runs.

6.2 Metrics Selection and Measurement

Before starting the real experiments, it is crucial to determine which metrics we will use to evaluate the model. For each parameter, it is also important to select a complete range of values in order to study the model with several configurations.

6.2.1 Quality metrics

To evaluate the quality of the model, two metrics have been used: the Normalized Mean Squared Error (NMSE) and the CPU time. The NMSE is based on the Mean Squared Error [41] (MSE), which represents the error between the real and predicted data (Equation 2.4). NMSE is computed by dividing the MSE by the variance of the true target values, as shown in Equation 6.3:

$$\text{NMSE} = \frac{\text{MSE}}{\text{Var}(\mathbf{t})} \qquad \text{with} \qquad \text{Var}(\mathbf{t}) = \frac{1}{N}\sum_{n=1}^{N}\left(t_n - \bar{t}\right)^2, \quad \bar{t} = \frac{1}{N}\sum_{n=1}^{N} t_n. \tag{6.3}$$

NMSE can also be calculated using R^2, as we observe in Equation 6.2.

$$\text{NMSE} = 1 - R^2. \tag{6.2}$$

NMSE represents the fraction of the target variance that remains unexplained by the model, while R^2 is the complementary fraction of variance explained. Concretely

- NMSE $= 0$ ($R^2 = 1$): all variance is explained; perfect fit.
- NMSE $= 0.25$ ($R^2 = 0.75$): 75 % of the variance is explained and 25 % remains unexplained.
- NMSE > 1 ($R^2 < 0$): the model performs worse than simply predicting the sample mean.

Additionally, we use CPU[11] time as a quality metric. This parameter provides a simple and intuitive way to compare the clustering and regularisation methods.

Finally, the presentation of the quality metrics is a crucial aspect of the experimental part. Therefore, we report each metric as the arithmetic mean accompanied by its standard deviation [42] (SD), using the compact notation *mean* ± *std*. Given m independent runs that yield values $(z_1, \dots, z_m)$, the SD is defined as

$$\mathrm{SD}(\mathbf{z}) = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(z_j - \bar{z})^2}, \tag{6.3}$$

where $\bar{z}$ denotes the arithmetic mean.

6.2.2 Model metrics

To compare the performance between different model configurations, we must define the model metrics. The ones that have been studied and discussed in this project are:

- **Number of neurons of the model:** In general, it would be calculated using the heuristic in Section 5.1.
- **Number of pruned neurons:** Lasso and AIC are regularisation techniques that reduce the number of neurons.
- **Shape parameter:** It is a parameter that has a strong impact on the performance model, so it should be studied in depth.
- **L2 regularisation parameter:** To find the optimal λ, it is mandatory to evaluate a good range of values.

[11]The CPU used is specified in Section A.2.2

6.3 Synthetic Experiment

In this experiment, we will evaluate the five regularisation methods using a one-variable function. The selected function is the *Doppler* signal introduced by Donoho and Johnstone [43].

$$f(x) = \sqrt{x(1-x)}\sin\left(\frac{2.1\pi}{x+0.05}\right). \tag{6.4}$$

The properties of this function make it a great choice for benchmarking in non-parametric regression studies. It has a non-stationary frequency, testing the model's ability to shrink its local scale where oscillations are rapid. Moreover, the function is infinitely differentiable on the interval $(0,1) \subset \mathbb{R}$. The function can be seen in Figure 15.

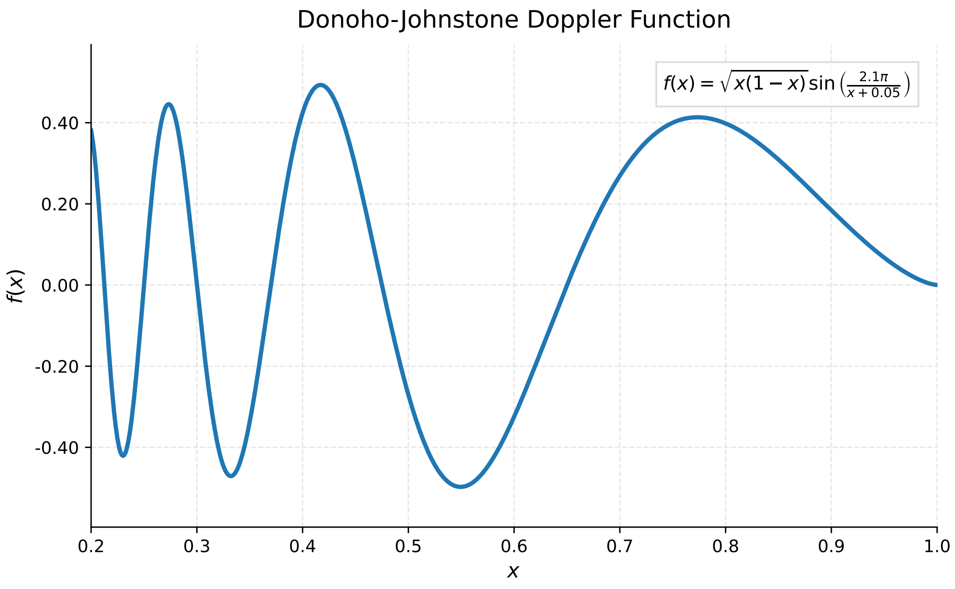


Figure 15: Doppler Function for interval $[0.2, 1.0]$.

Before starting the experiment, we will define the problem metrics that will be used. For this experiment, we will consider the size of the training and testing data, the interval for the data points, and the amount of Gaussian noise applied.

Parameter	**Description**
Total Samples N	Total number of generated data points.
Train/Val/Test Split	Percentage of data used for each phase. $\{70\%, 15\%, 15\%\}$
x-interval $[a, b]$	Range from which x values are sampled. $[0.2, 1.0]$
Gaussian noise σ	Gaussian noise on targets.

Table 5: Problem metrics for Doppler experiment.

The Synthetic Experiment can be divided into two parts. First of all, we compare our clustering method against the common clustering method for RBFNN. In the second part, we evaluate the regularisation techniques.

6.3.1 Comparison of Clustering Algorithms

In the first part of the synthetic experiment, we compare hierarchical clustering (our implementation) against K-means (the most common implementation) for selecting the number of neurons following the heuristic: *select the approximate minimum number of neurons that causes the model to overfit.*

The main objective of this comparison is to contrast the CPU time for each clustering method and identify any differences in the number of neurons.

N	σ	**NMSE**	**# Neurons**	**CPU Time (s)**
500	0.05	0.03 ± 0.01	78.4 ± 18.01	4.01 ± 1.90
	0.10	0.11 ± 0.02	73.1 ± 15.59	4.00 ± 2.45
	0.20	0.35 ± 0.05	83.9 ± 12.98	4.73 ± 3.09
	0.30	0.55 ± 0.07	78.1 ± 14.26	4.45 ± 2.50
1000	0.05	0.02 ± 0.01	115.7 ± 24.3	22.11 ± 11.38
	0.10	0.10 ± 0.01	135.2 ± 29.51	24.48 ± 12.76
	0.20	0.31 ± 0.04	157.8 ± 22.02	26.48 ± 13.11
	0.30	0.50 ± 0.05	150.2 ± 27.48	27.15 ± 12.19

Table 6: Hierarchical clustering results depending on $N = [500, 1000]$ (number of data points) and $\sigma = [0.05, 0.10, 0.20, 0.30]$ (Gaussian noise). NMSE, number of neurons, and CPU time are reported as mean ± std (10 runs).

N	σ	**NMSE**	**# Neurons**	**CPU Time (s)**
500	0.05	0.03 ± 0.01	75.9 ± 10.0	106.17 ± 31.90
	0.10	0.11 ± 0.02	72.2 ± 13.6	105.64 ± 32.15
	0.20	0.35 ± 0.05	90.2 ± 12.2	118.76 ± 33.99
	0.30	0.57 ± 0.08	81.9 ± 11.2	114.61 ± 33.57
1000	0.05	0.03 ± 0.01	125.0 ± 27.2	288.13 ± 61.02
	0.10	0.10 ± 0.02	126.4 ± 25.1	236.23 ± 73.45
	0.20	0.31 ± 0.04	152.8 ± 23.4	251.36 ± 69.63
	0.30	0.51 ± 0.05	159.8 ± 29.8	249.81 ± 67.19

Table 7: K-means results depending on $N = [500, 1000]$ (number of data points) and $\sigma = [0.05, 0.10, 0.20, 0.30]$ (Gaussian noise). NMSE, number of neurons, and CPU time are reported as mean ± std (10 runs).

From the Tables 6 and 7, we observe that both methods achieve comparable Normalized Mean Squared Error (NMSE) and select a similar number of neurons N. Additionally, these values appear to be independent of the chosen method but dependent on the number of data points and Gaussian noise. However, there is a noticeable difference in CPU time between methods. In fact, for a data set with $N = 500$, our approach to finding the number of neurons is approximately 96% faster than the k-means method and for a $N = 1000$

is approximately 90%. Furthermore, the run-time standard deviation of the hierarchical method is much more consistent than that of k-means across multiple runs.

6.3.2 Comparison of Regularisation Techniques

In the second part of the synthetic experiment, we focus on comparing the five regularisation methods: Ridge, Lasso, AIC, Sequential AIC + Ridge and Sequential AIC + Ridge. In addition, we have three different strategies for Ridge Regression: GCV loop, iterative lambda and RidgeCV. As there are too many techniques to compare, the synthetic experiment is divided into two parts.

In the first part, we analyse the three implementations of Ridge Regression, described in Section 5.3.2, with the aim of identifying the most suitable one for use in the second part.

Ridge Method	σ	**NMSE**	λ	**CPU Time (s)**
GCV loop	0.05	0.03 ± 0.01	1^{-3}	0.13 ± 0.03
	0.10	0.11 ± 0.02	1^{-2}	0.10 ± 0.02
	0.20	0.34 ± 0.05	3.8^{-2}	0.15 ± 0.03
	0.30	0.57 ± 0.08	3.1^{-2}	0.14 ± 0.03
Iterative lambda	0.05	0.04 ± 0.01	1^{-4}	0.12 ± 0.03
	0.10	0.14 ± 0.04	2.3^{-3}	0.11 ± 0.04
	0.20	0.37 ± 0.11	2.4^{-3}	0.13 ± 0.04
	0.30	0.59 ± 0.09	7.9^{-3}	0.14 ± 0.04
RidgeCV	0.05	0.03 ± 0.01	1^{-3}	0.11 ± 0.04
	0.10	0.11 ± 0.02	1.5^{-2}	0.11 ± 0.04
	0.20	0.35 ± 0.05	2.5^{-2}	0.12 ± 0.05
	0.30	0.58 ± 0.09	5.2^{-2}	0.12 ± 0.05

Table 8: Comparison of the three Ridge regression strategies with $N = 500$ and $\sigma = [0.05, 0.10, 0.20, 0.30]$. NMSE, regularisation parameter and CPU time are reported. The mean $\pm$ std is computed for 10 runs.

According to the metrics in Table 8, the three methods behave very similarly. They achieve almost identical CPU times. However, there are differences in the regularisation parameter (λ). However, since all the values are very small, these differences do not have a significant impact. Furthermore, as Gaussian noise increases, the value of lambda also increases, indicating that more regularisation is needed in the model. Generally, low values of lambda indicate that little regularisation is needed. This makes sense; therefore, we use a low N for the experiment.

Now, in order to select one of the three methods for the second part, we will analyse the NMSE metric. The following bar plot has been generated.

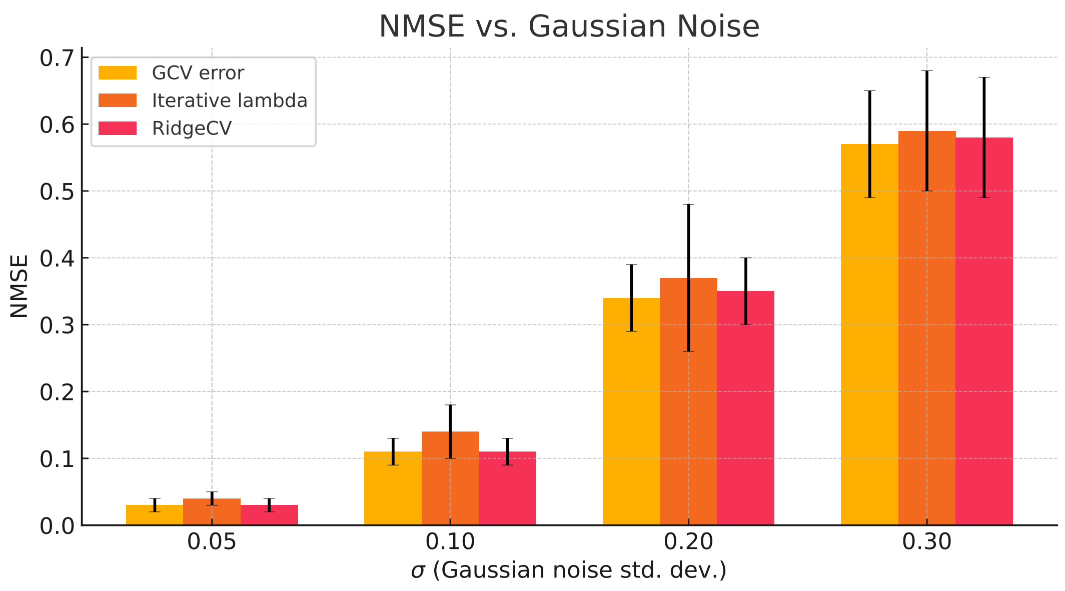


Figure 16: Plot for NMSE against Gaussian Noise for the three Ridge regression strategies with $N = 500$. The black line illustrates the standard deviation.

We observe an almost identical NMSE for the GCV loop and RidgeCV methods, with a similar standard deviation. The iterative lambda method produces slightly worse NMSE and its standard deviation is also slightly worse. In summary, our two Ridge Regression implementations produce similar results to the RidgeCV [34] method.

Finally, the GCV loop strategy obtains more consistent results than the iterative lambda strategy as its standard deviation is low. We therefore decided to use GCV loop for the second part of the experiment.

Once the ridge method has been decided upon, we will continue with the second part of the synthetic experiment. In this part, we contrast the results of all the strategies presented in Section 5.3.

Method	σ	NMSE	L1	L2 λ	# Neurons	Radius
Ridge	0.05	0.06 ± 0.01	-	1^{-5}	79/79	0.07
	0.10	0.11 ± 0.02	-	0.07	79/79	0.03
	0.20	0.34 ± 0.06	-	0.27	79/79	0.03
	0.30	0.57 ± 0.08	-	0.79	79/79	0.03
Lasso	0.05	0.09 ± 0.03	5.6^{-5}	-	25/79	0.03
	0.10	0.13 ± 0.03	5.7^{-5}	-	32/79	0.03
	0.20	0.36 ± 0.07	3.8^{-3}	-	29/79	0.03
	0.30	0.59 ± 0.08	7.1^{-4}	-	27/79	0.03
AIC	0.05	0.03 ± 0.01	-	-	20/79	0.07
	0.10	0.11 ± 0.02	-	-	21/79	0.07
	0.20	0.35 ± 0.06	-	-	21/79	0.07
	0.30	0.60 ± 0.09	-	-	22/79	0.07
Lasso+Ridge	0.05	0.09 ± 0.04	5.6^{-5}	0.10	25/79	0.03
	0.10	0.12 ± 0.04	5.7^{-5}	0.12	32/79	0.03
	0.20	0.35 ± 0.06	3.8^{-3}	0.22	29/79	0.03
	0.30	0.57 ± 0.08	7.1^{-4}	0.39	27/79	0.03
AIC+Ridge	0.05	0.05 ± 0.01	-	1^{-5}	21/79	0.07
	0.10	0.14 ± 0.03	-	2.9^{-3}	44/79	0.03
	0.20	0.35 ± 0.06	-	7.2^{-2}	45/79	0.03
	0.30	0.59 ± 0.09	-	1^{-5}	22/79	0.07

Table 9: Results of the regularisation methods for $N = 500$ and $\sigma = [0.05, 0.10, 0.20, 0.30]$. NMSE, regularisation parameter for Ridge, CPU time, total number of neurons, number of pruned neurons and shape parameter are compared. The mean $\pm$ std is computed for 10 runs.

In Table 9, several metrics are presented. First of all, we examine the number of neurons pruned by Lasso and AIC. AIC is more aggressive cutting neurons. Additionally, both methods eliminate more neurons in the presence of low Gaussian noise and conserve more neurons when the noise level is increased. Furthermore, we detect a correlation between radius and the number of pruned neurons. In the AIC + Ridge method, when the radius is **0.03**, an average of **58.5** neurons are eliminated. In contrast, when the radius is **0.07**, **34.5** neurons are eliminated on average. Overall, increasing the radius from 0.03 to 0.07 yields a reduction of 41.0% in pruned neurons. This is because, with a low radius, each neuron has less scope, so more neurons are needed to compensate.

The Normalised Mean Squared Error (NMSE) varies depending on the amount of Gaussian noise present, but not on the regularisation method used. All five methods produce similar results with 500 data points. However, this appears to be too few data points to observe any differences in the NMSE, so a larger dataset is used in the next experiment.

Method	$\sigma = 0.05$	$\sigma = 0.10$	$\sigma = 0.20$	$\sigma = 0.30$
Ridge	0.03 ± 0.00	0.06 ± 0.02	0.10 ± 0.03	0.11 ± 0.03
Lasso	0.20 ± 0.05	0.42 ± 0.12	0.49 ± 0.14	0.57 ± 0.23
AIC	0.45 ± 0.14	0.42 ± 0.13	0.46 ± 0.18	0.45 ± 0.19
Lasso+Ridge	0.24 ± 0.01	0.46 ± 0.20	0.52 ± 0.13	0.60 ± 0.19
AIC+Ridge	0.49 ± 0.15	0.46 ± 0.15	0.49 ± 0.18	0.48 ± 0.19

Table 10: Mean CPU time (in seconds) for each regularisation method at noise levels $\sigma \in \{0.05, 0.10, 0.20, 0.30\}$, shown as mean±std over 10 runs.

As illustrated in Figure 17, the largest performance gap between the methods is in CPU time. Ridge is clearly the fastest method. For low levels of Gaussian noise, AIC performs the worst. However, as the level of Gaussian noise increases, the models perform similarly. This is surprising, given that AIC uses the greedy backward stepwise selection algorithm. For this reason, we had expected Lasso to be faster than AIC.

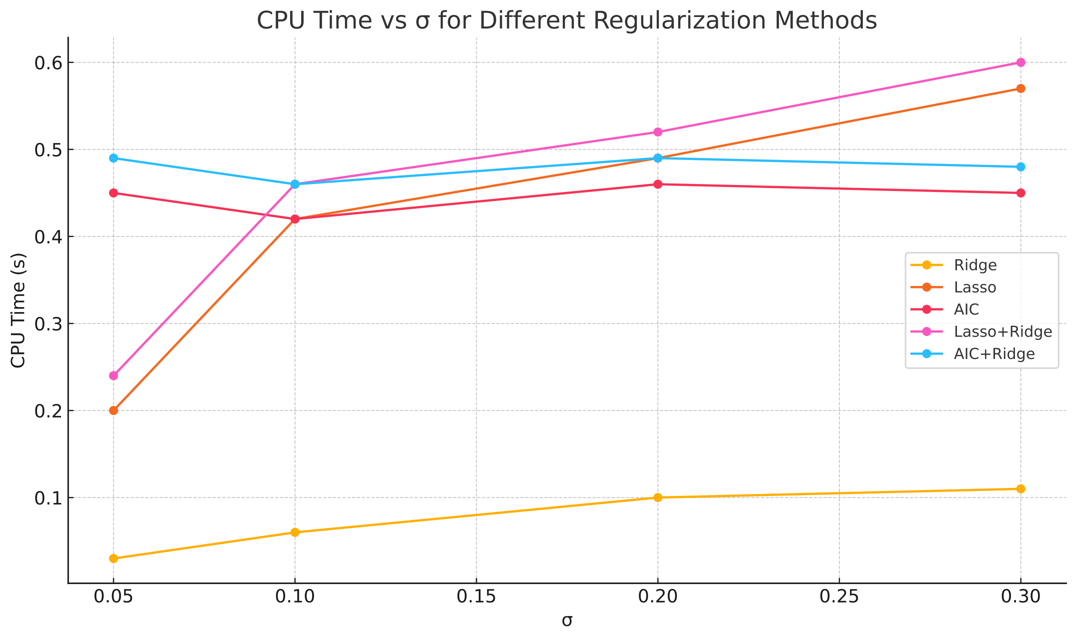


Figure 17: CPU time comparison for different regularisation methods under varying Gaussian noise levels.

6.4 Realistic Experiment

In this experiment, we will evaluate our training method with a real dataset. The chosen dataset is called *Concrete Compressive Strength* [44]. The Concrete Compressive Strength, donated by I-Cheng Yeh in 2007, contains 1030 laboratory-measured concrete mixes. It is formed by 8 quantitative input variables and 1 quantitative output variable:

- **Cement (component 1)** – quantitative, measured in $\mathrm{kg\,m^{-3}}$ of mixture (input variable).

- **Blast Furnace Slag (component 2)**: quantitative, kg m^{-3} (input variable).
- **Fly Ash (component 3)**: quantitative, kg m^{-3} (input variable).
- **Water (component 4)**: quantitative, kg m^{-3} (input variable).
- **Superplasticizer (component 5)**: quantitative, kg m^{-3} (input variable).
- **Coarse Aggregate (component 6)**: quantitative, kg m^{-3} (input variable).
- **Fine Aggregate (component 7)**: quantitative, kg m^{-3} (input variable).
- **Age**: quantitative, days (1–365) (input variable).
- **Concrete Compressive Strength**: quantitative, megapascals (MPa) (output variable / target).

We selected the Concrete Compressive Strength dataset because several studies have used this dataset to evaluate the performance of machine learning models. Notable examples include the work of Yeh [45], who originally introduced the dataset, as well as more recent studies such as *Learning Linear Feature Space Transformations in Regression Problems Using CMA-ES* [46], which uses our exact data set, and *High correlated variables creator machine: Prediction of the compressive strength of concrete* [47], which only uses 516 data points. We utilize these three studies for comparison purposes, since they use Normalized Mean Squared Error (NMSE) or R^2, as does our method. However, none of these studies provide any time measurement for their models.

Method	**Total**	**Selected**	**Pruned**	**CPU Time (s)**	λ
Ridge	232	232	0	0.53 ± 0.11	5.2×10^{-2}
Lasso	232	99.6 ± 3.86	132.4 ± 3.86	2.79 ± 0.51	–
AIC	232	153.1 ± 8.08	78.9 ± 8.08	33.35 ± 8.97	–
Lasso+Ridge	232	99.6 ± 3.86	132.4 ± 3.86	3.10 ± 0.56	0.014
AIC+Ridge	185	128.9 ± 5.40	56.1 ± 5.40	30.56 ± 2.08	1×10^{-3}

Table 11: Results of the regularisation methods for Concrete Compressive Strength dataset. The total number of neurons, selected neurons, pruned neurons, CPU time and regularisation parameter are compared.

From Table 11, we can observe the evolution of neurons depending on the model. Lasso eliminates 57% of the total neurons and AIC eliminates approximately 30%. Lasso is undoubtedly more aggressive in pruning neurons than AIC. However, AIC attains a lower NMSE than Lasso, despite keeping a larger fraction of neurons.

Regarding the CPU time, Ridge obtains the lowest time, followed by Lasso and the sequential Lasso + Ridge. The final ones are AIC and sequential AIC + ridge, which are significantly behind the rest.

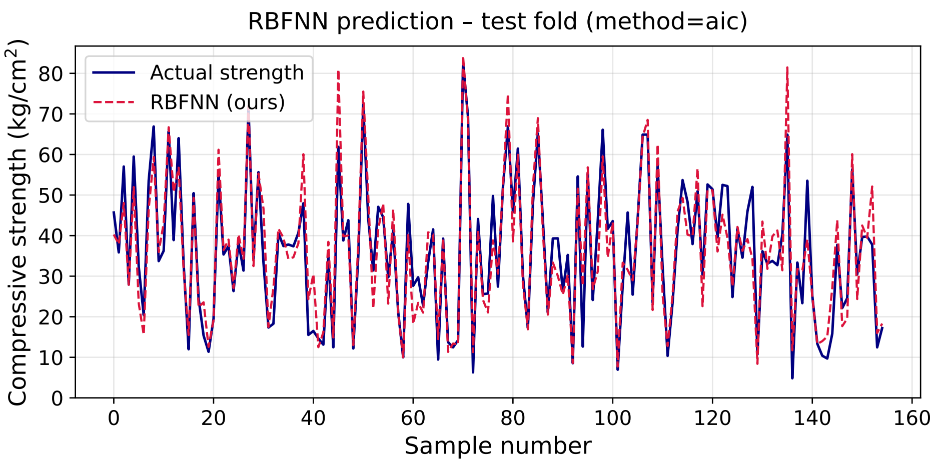
Source	**Model**	**Testing NMSE**
This work	Ridge	**0.166 ± 0.032**
	Lasso	**0.232 ± 0.039**
	AIC	**0.170 ± 0.034**
	Lasso + Ridge	**0.209 ± 0.033**
	AIC + Ridge	**0.143 ± 0.029**
Original reference [45]	Best Model	**0.078**
	Worst Model	**0.186**
Reference [46]	Best Model	**0.141**
	Worst Model	**0.156**
Reference [47]	Best Model	**0.084**
	Worst Model	**0.189**

Table 12: Testing-set NMSE of our five RBFNN methods compared with the best (green) and worst (red) NMSE reported in two related studies.

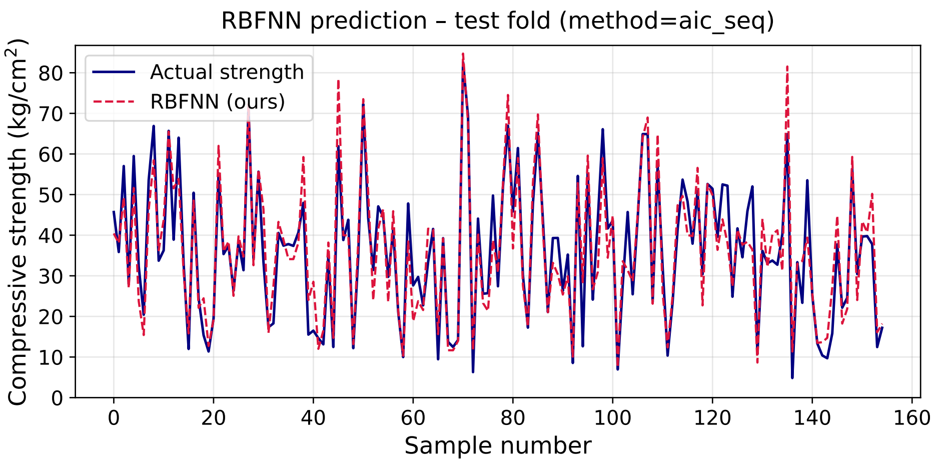
Table 12 summarizes the Normalized Mean Squared Error (NMSE) obtained by our model and by two recent studies. The **best model** is represented by a green color and the **worst model** by a red color. Our best method is Sequential AIC + Ridge, achieves an NMSE of 0.143 ± 0.029 on the dataset, which is similar to the 0.141 reported by Žegklitz and Posík [46], who analysed the same dataset using the computationally intensive CMA-ES. However, we do not know how they split the data for training and testing. Shishegaran [47] quotes a lower best NMSE of 0.084, but this figure comes from a much smaller sample size of 516 mixes. Finally, the original paper [45] achieves 0.078, which is the best NMSE.

Taken together, these results show that our two-stage AIC + Ridge approach achieves state-of-the-art accuracy across the entire dataset while remaining significantly simpler.

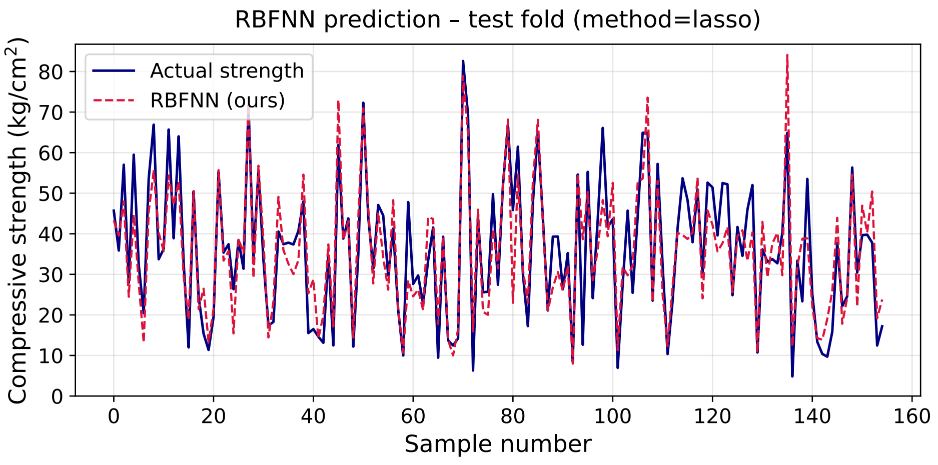
Finally, we plot the prediction for each method and the true values of the target.



(a) AIC

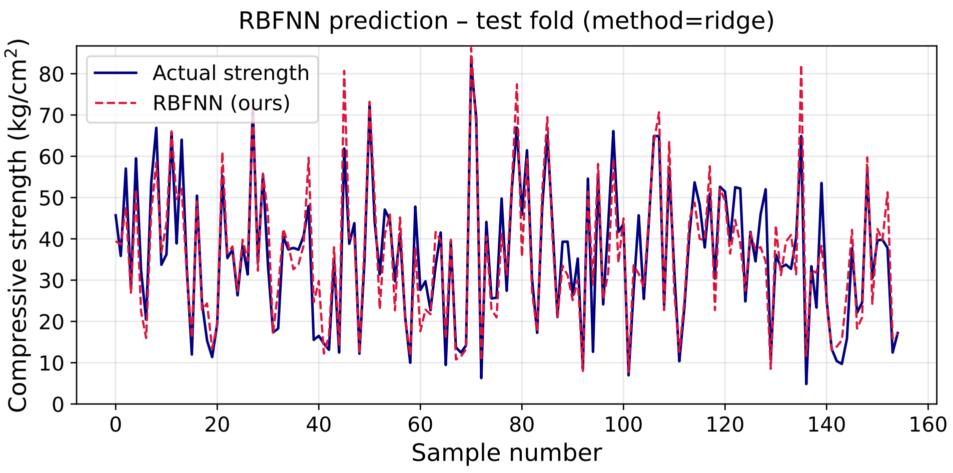


(b) AIC + Ridge

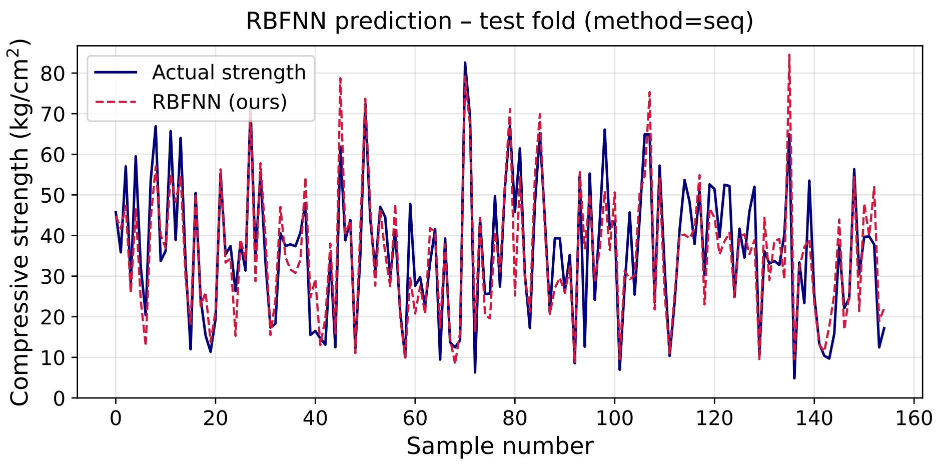


(c) Lasso

Figure 18: Predicted versus actual values: AIC, AIC + Ridge, and Lasso.



(a) Ridge



(b) Lasso + Ridge

Figure 19: Predicted versus actual values: Ridge and Lasso + Ridge.

6.5 Scalability Evaluation for Ridge

Both of the previous experiments have been conducted using datasets of the order of 10^3, and we have observed that our ridge method obtains fairly accurate results with an impressive CPU time. For that reason, we will conduct a scalability test to determine if the incredible CPU time is maintained for large datasets and the accuracy does not decrease significantly.

For this experiment, we use the dataset called *Physicochemical Properties of Protein Tertiary Structure* [48], which contains 45730 instances and the variables are:

Variable	**Description**
RMSD	Size of the residue
F1	Total surface area
F2	Non-polar exposed area
F3	Fractional area of exposed non-polar residue
F4	Fractional area of exposed non-polar part of residue
F5	Molecular mass–weighted exposed area
F6	Average deviation from standard exposed area of residue
F7	Euclidean distance
F8	Secondary structure penalty
F9	Spatial distribution constraints (N, K value)

Table 13: Explanation of variables in the physicochemical protein dataset.

Let $D = \{x_1, x_2, \ldots, x_N\}$ denote our full dataset of size N. To evaluate how performance and computational cost scale with problem size, we construct sub-datasets by truncating D at various block sizes:

$$B = \{\, b_1, b_2, \ldots, b_K\} = \{5\,000,\ 10\,000,\ 25\,000\},$$

we then run our entire processing pipeline for each b_i. All experiments rely on a **70/15/15** split: 70 % of each data set for training, 15 % for validation, and the remaining 15 % for testing.

Samples	Neurons	r	NMSE$_{\text{test}}$	CPU time (s)	Ridge Param
5000	1048.0 ± 130.5	1.4962 ± 0.4475	0.5976 ± 0.0118	183.17 ± 29.27	1^{-2}
10000	2358.0 ± 304.4	1.0136 ± 0.3123	0.5350 ± 0.0112	400.35 ± 65.87	2.1^{-1}
25000	4562.4 ± 512.4	0.8776 ± 0.2430	0.4802 ± 0.0162	1256.78 ± 257.09	5.3^{-1}

Table 14: Summary evaluation metrics for the scalability test: neurons, shape parameter, test NMSE, training time, and ridge regularisation parameter. (mean ± standard deviation for 10 runs).

First, we observe that the number of neurons grows approximately linearly with each sample, as the proportion obtained by dividing the mean number of neurons, m, by the number of samples, p, is approximately 0.20 for the three blocks. Specifically, $1048/5000 \approx 0.21$, $2358/10000 \approx 0.24$, $4556.4/25000 \approx 0.18$.

Secondly, the shape parameter decreases from $r = 1.50$ to 0.88, reflecting the shorter average neighbour distance in denser data. Crucially, the test error $\text{NMSE}_{\text{test}}$ decreases from 0.598 ± 0.012 (5 k samples) to 0.480 ± 0.016 (25 k samples), indicating that the additional variance introduced by extra centers is more than compensated by the variance reduction due to a larger training set. Moreover, as the number of neurons increases, the ridge parameter λ must also increase (i.e. stronger regularisation) to keep the effective degrees of freedom in check.

Finally, although CPU time increases, it remains manageable, suggesting that models of up to $\sim$25 k samples are tractable on commodity hardware and that larger sets can also be handled.

7 Conclusion

In this project, we focused on the design, implementation, and testing of a novel method for training Radial Basis Function Neural Networks (RBFNNs) using hierarchical clustering for neuron selection and output weight computation. Furthermore, the weights are calculated using different regularisation techniques: Ridge, Lasso, AIC, sequential Lasso + Ridge and sequential AIC + Ridge.

The software was developed from scratch in Python to give users the option to choose between all the regularization methods. Although the source code is not the central focus of this thesis, it will be publicly available on GitHub for future work.

The correct functioning of the model has been validated by reproducing Orr's synthetic example. Moreover, a synthetic dataset was employed to demonstrate that using hierarchical clustering for our method and heuristic is significantly faster than K-means. Additionally, since hierarchical clustering determines the number of neurons automatically, our approach eliminates the need to pre-specify the number of clusters—a limitation that K-means inherently carries.

Furthermore, an evaluation on a real dataset was conducted to assess a practical application of the proposed method. Comparing the five regularisation methods has yielded interesting results, including an observed superior performance by the sequential AIC + Ridge, achieving near-state-of-the-art accuracy with minimal computational cost.

Finally, a scalability test on a large real dataset was performed. Although its gathered NMSE does not match state of the art in the existing literature, the model achieves respectable accuracy while requiring only minimal computation. This favorable accuracy-vs-cost trade-off makes it a solid, computationally efficient option for large datasets that can run comfortably on low-power hardware.

7.1 Future Work

There are several avenues worth exploring. Firstly, the focus of this project is exclusively on regression problems. A logical next step would be to consider classification problems. Secondly, the shape parameter is estimated globally for the RBFNN. Investigating a locally adaptive or density-aware method for calculating the shape parameter would enable each basis function to align with the scale of its neighborhood and adapt more effectively to varying cluster densities. Thirdly, the regularisation parameter for analytic Ridge regression is estimated globally. However, it is feasible to employ a different regularisation parameter for each basis function, providing a mechanism to adapt smoothness to local conditions. This approach would allow each regularisation parameter to be optimized.

A Project Management

A.1 Methodology and Rigor

A.1.1 Methodology

The project will follow a structured methodology based on the following steps:

1. **Literature Review:** Analysis of previous studies on RBF networks, clustering techniques, and regularisation strategies to establish a solid reference framework.

2. **Method Implementation:** Development of the training algorithm that integrates hierarchical clustering with weight adjustment using different regularisation methods.

3. **Experimentation and Testing:** Application of the proposed method to various datasets to assess its effectiveness compared to traditional approaches.

4. **Performance Evaluation:** Measurement of error rates and the model's generalization capability.

5. **Optimization and Fine-Tuning:** Refinement of the model through hyperparameter tuning and exploration of improvements in regularisation.

6. **Documentation and Analysis:** Detailed recording of experiments and analysis of results to ensure study replicability.

These tasks will be managed using the Kanban method. Tasks will be broken down into smaller steps and represented as cards on a Kanban board divided into columns reflecting different workflow states (To Do, In Progress, Under Review and Completed).

Additionally, sprint-based iterations will be implemented to allow prioritisation of the most critical tasks.

A.1.2 Monitoring tools and validation

To ensure the quality and reproducibility of the study, the following tools and methodologies will be employed:

- **Version Control:** GitHub will be used to manage code development and maintain a complete history of changes.

- **Development Environment:** Implementation in Notebooks to facilitate experimentation and result analysis.

- **Error Analysis and Hyperparameter Tuning:** Application of hyperparameter search techniques and model selection based on statistical validation.
- **Detailed Documentation:** A structured record of each experiment and its results will be maintained to ensure the study's reproducibility.

A.2 Project Planning

The starting date of this project aligns with the beginning of the project management deliverables, which is February 19, 2025. On the other hand, the submission deadline for the written report is June 18, 2025. The defense is on June 26, 2025.

Taking all this into account, there are approximately 95 working days available (excluding holidays and Sundays). It is planned to work 4 hours approximately every day.

A.2.1 Task Definition

Next, all the tasks to be addressed in the project will be defined. First, the tasks related to project management and planning will be outlined:

- **Definition of the work environment:** Choosing the appropriate work environment is a fundamental part of any project. It is necessary to evaluate the advantages and disadvantages of the different technologies that can be used.
- **Context and scope:** It is important to define a clear and feasible objective, providing context and justification for the chosen approach.
- **Project planning:** Proper planning is essential to complete the work on time. Specifically, breaking the project into smaller tasks ensures that each aspect receives the necessary attention.
- **Budget and sustainability:** Understanding the economic impact of a project is crucial. Defining a budget and adhering to it is key.
- **Final project definition:** This involves analyzing previous tasks, making adjustments, and applying improvements.
- **Progress tracking:** To maintain effective control over the project's progress, all work done each week will be recorded in a template. This will also serve to keep the tutor informed and maintain a structured log.
- **Meetings:** Both online and in-person meetings will be held with the tutor to discuss the project's progress. Generally, meetings will take place biweekly.

This project has a significant research component. Therefore, before starting the experimental phase, it is mandatory to review key concepts and existing studies in the field. The research phase can be divided into several blocks:

- Research in the RBFNs area
- Research in hierarchical clustering
- Research in regularisation techniques and overfitting prevention

The following tasks are related to the implementation of the proposed method. These tasks rely on prior research, though some can be performed in parallel.

- Programming the Radial Basis Function Network will be divided into two subtasks:
  - Implementation of hierarchical clustering
  - Implementation and testing of different regularisation methods

Once the RBFN is implemented, the experimental phase will begin. This stage ensures the correctness of the method by testing each function's proper functionality.

- Test the correct functioning: Throughout the programming phase, small tests will be conducted to segment the process and verify each step.
- Obtain the datasets for experimentation.
- Experiment with the implemented methods and datasets. Some methods may require parameter tuning.
- Analyse the experimental results and draw conclusions.

Lastly, all the findings and results obtained throughout the project will be compiled into a structured written report. Additionally, the oral defense of the thesis will be prepared.

ID	NAME	TIME (hours)	DEPENDENCIES	RESOURCES
T1	**Project management**	**120**		
T1.1	Definition of the work environment	**5**		PC, R
T1.2	Context and scope	**20**	T2*	PC, R, GEPT
T1.3	Project planning	**15**		PC, R, GEPT
T1.4	Budget and sustainability	**15**	T1.3	PC, R, GEPT
T1.5	Final project definition	**20**	T1.2, T1.3, T1.4	PC, R, GEPT
T1.6	Progress tracking	**20**		PC, R, T
T1.7	Meetings	**25**		PC, R, T
T2	**Research**	**140**		PC, R, papers, books
T2.1	Research in RBFNs area	**55**		PC, R, papers, books
T2.2	Research in clustering jerárquico	**40**		PC, R, papers, books
T2.3	Research in regularisation and overfitting prevention	**45**		PC, R, papers, books
T3	**Programming**	**55**		PC, R, Python, git
T3.1	Implementation of hierarchical clustering	**25**	T2.2*	PC, R, Python, git
T3.2	Implementation of regularisation	**30**	T2.3*	PC, R, Python, git
T4	**Experiments and analysis**	**120**	T3	PC, R, Python, git
T4.1	Test the correct functioning	**30**	T3	PC, R
T4.2	Obtain the datasets for experimentation	**10**		PC, R, T
T4.3	Experiment with the implemented methods and datasets	**50**	T3, T4.2	PC, R
T4.4	Analyse and draw conclusions	**30**	T4.3	PC, R, T
T5	**Project documentation**	**70**	T4.4*	PC, R
T6	**Bachelor thesis defense preparation**	**25**	T5	PC, R, T
	TOTAL	**530**		

Table 15: Summary of the information of the tasks. [Own creation]
*: The task that creates the dependency does not have to end to be able to begin the task.
GEPT: GEP tutor T: Tutor R: Researcher

A.2.2 Resources

Managing project resources effectively is essential to achieving a high-quality final product. Specifically, resources have been categorized into four types: human, hardware, software, and material resources.

Human Resources

This project involves three human resources. The researcher is responsible for developing the project, which includes planning, conducting experiments, analyzing the results and documenting the entire process. The tutor will guide the researcher throughout the project's development, providing support and direction. Finally, the GEP tutor will assist the researcher with project management during the first month.

Hardward Resources

The primary resource used in this project will be a computer. Specifically, the following device will be used:

- **Laptop:** Lenovo ThinkPad T14s Gen 1
- **Processor:** AMD Ryzen 7 Pro
- **RAM:** 16 GB

Additionally, other connectivity resources, such as the router, will also be considered.

Software Resources

Several types of software will be required for this project. The Gantt chart 20 will be created using GanttProject. Information logging and version control will be managed through a GitHub repository. The development environment will be Visual Studio Code, with Python as the programming language. Overleaf will be used for documentation and project management, while Google Meet will be the platform for online meetings with the thesis supervisor.

Material Resources

In projects with a significant research component, it is crucial to thoroughly review previous studies in the field. This will require reading a large number of research papers and books.

A.2.3 Risk Management: Alternatives Plans

During the execution of the project, various issues may arise. A key aspect of mitigating them is the ability to anticipate and effectively counteract potential risks.

- **Inexperience in the programming language [Low risk].** The programming language will not be an issue since one has been chosen in which the researcher has prior experience. If a required library is unfamiliar, a new task will be created to focus on understanding and mastering that library.

- **Project deadline [High risk].** Poor estimation of task durations may result in failure to meet established deadlines. If this occurs, the project plan will need to be restructured to meet deadlines, potentially increasing the number of working hours per day.

- **Impossibility of meeting with the supervisor in person [Extreme risk].** Given the highly technical nature of this project, the guidance of the thesis supervisor is crucial for achieving a high-quality final result. If in-person meetings are not possible, online meetings will be arranged to ensure the continuous flow of information between both parties.

- **Serious illness or injury [Low risk].** Although unlikely, there is a possibility of experiencing a critical illness or injury that could delay or prevent the completion of certain tasks. In such a case, the project plan would need to be restructured, and in an extreme scenario, the project scope might have to be reduced.

- **Technological failure [Medium risk].** If the primary hardware resource fails or breaks, a replacement would be required, as only one computer is available. This would entail an additional financial cost, affecting the project's budget.

A.2.4 Gantt Chart

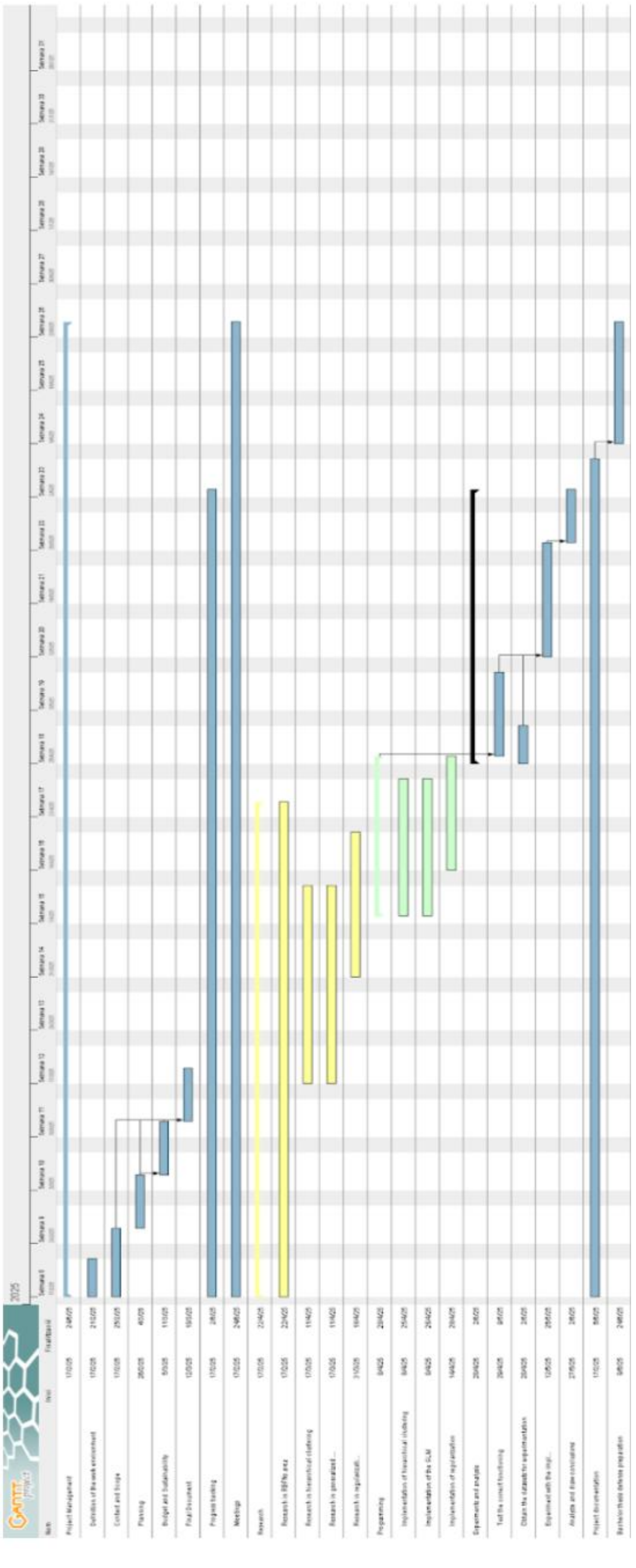


Figure 20: Gantt chart (rotated).

A.3 Budget and Sustainability

It will be described the elements to consider when doing the budget estimation, which includes personnel costs per task, generic costs and other costs. Moreover, it will be defined management control mechanisms to control the deviations that can appear in the project due to unforeseen obstacles. Finally, some aspects of sustainability will be discussed.

A.3.1 Budget

Personnel cost per activity

In this section the total cost of each task defined in section 2.4 is calculated. The total cost will be calculated by adding up the cost of the work for each personnel. To get the cost for each worker, we will look at how many hours he has worked and multiply it by the cost per hour. In this project there are 4 types of workers: the project manager is responsible for the planning and the correct development of the project. Secondly, the programmer will be in charge of the programming and the code testing part. On the other hand, the researcher has to experiment, analyse the results and draw conclusions. Finally, the technical writer has to document everything in order to write the project report. All the functions will be carried out by me, although the tutor and the GEP tutor will also be involved in the role of project manager. The salary calculation will be based on Table 16 and all calculations are shown in Table 17.

Role	**Annual Salary (€)**	**Price per hour (€)**	**Total including SS (€)**	**Role played by**
Project manager	35.000	17,95	47.250	GEPT, R, T
Programmer	30.000	15,38	40.500	R
Researcher	31.141	15,97	42.040,35	R
Technical writer	40.000	20,51	54.000	R

Table 16: Annual salary of the different project roles. [Own creation]
GEPT: GEP tutor | **T**: Tutor | **R**: Researcher

ID	NAME	Total hours	Project Manager	Programmer	Researcher	Tech. Writer	Cost (€)
T1	**Project management (120 h)**						
T1.1	Definition of the work environment	5,00	5,00	0,00	0,00	0,00	121,16
T1.2	Context and scope	20,00	20,00	0,00	0,00	0,00	484,65
T1.3	Project planning	15,00	15,00	0,00	0,00	0,00	363,48
T1.4	Budget and sustainability	15,00	15,00	0,00	0,00	0,00	363,48
T1.5	Final project definition	20,00	20,00	0,00	0,00	0,00	484,65
T1.6	Progress tracking	20,00	20,00	0,00	0,00	20,00	1038,42
T1.7	Meetings	25,00	25,00	25,00	25,00	25,00	2356,08
T2	**Research (140 h)**						
T2.1	Research in RBFNs area	55,00	0,00	0,00	55,00	0,00	1185,77
T2.2	Research in clustering jerárquico	40,00	0,00	0,00	40,00	0,00	862,38
T2.4	Research in regularisation and overfitting prev.	45,00	0,00	0,00	45,00	0,00	970,17
T3	**Programming (55 h)**						
T3.1	Implementation of hierarchical clustering	25,00	0,00	25,00	0,00	0,00	519,08
T3.2	Implementation of regularisation	30,00	0,00	30,00	0,00	0,00	622,89
T4	**Experiments and analysis (120 h)**						
T4.1	Test the correct functioning	30,00	0,00	0,00	30,00	0,00	646,78
T4.2	Obtain the datasets for experimentation	10,00	0,00	0,00	10,00	0,00	215,59
T4.3	Experiment with implemented methods	50,00	0,00	0,00	50,00	0,00	1077,97
T4.4	Analyse and draw conclusions	30,00	0,00	0,00	30,00	0,00	646,78
T5	**Project documentation (70 h)**						
T6	**Bachelor thesis defense preparation (25 h)**						
T6	Bachelor thesis defense preparation	25,00	25,00	0,00	0,00	0,00	605,81
	TOTAL	530,00	155,00	80,00	275,00	110,00	14 226,50

Table 17: Personnel cost for each task defined. [Own creation]

Generic cost

Amortization

One key aspect to take into account is the amortization of the resources used in the project. The lifespan of a laptop is usually 5 years. The equation to compute the amortization for each resource is the following:

$$\text{Amortization (€)} = \text{Resource price} \times \frac{1}{\text{lifespan}} \times \frac{1}{\text{days worked}} \times \frac{1}{\text{hours worked per day}} \times \text{hours used}$$

In this project, all the software resources are free to use, so we will not include them in the equation. The amortization costs are shown below in Table 18.

Hardware	**Price (€)**	**Amortization (€)**
ThinkPad T14s	1499	397,24

Table 18: Amortization costs for the hardware resources. [Own calculations]

Electric cost

The actual cost of the kWh is 0,1400 €. Now, in order to obtain the total cost, we multiply by the hours that we have the computer on. Table 19 shows the results.

Hardware	**Power (W)**	**Time used (h)**	**Consumption (kWh)**	**Cost (€)**
ThinkPad T14s	65	530	74,2	4,82

Table 19: Electric cost of the hardware resources used in the project. [Own calculations]

Internet cost

The internet rate cost 44 € per month. Taking into account that the project lasts 4 months and that we work 4 hours per day, the internet costs are:

$$44 \times \frac{€}{\text{month}} \times 4 \text{ months } \times \frac{4 \text{ hours}}{24 \text{ hours}} = 29,3£$$

Travel cost

For meetings with the tutor I take public transport every week. The T-Jove allows me to make as many journeys as I want for 3 months. As the job lasts about 4 months I have to buy two cards at a price of 40 € each. The total cost is 80 €.

Generic cost of the project

Table 20 shows all the costs introduced in the preceding sections. The result will correspond to the CG cost (generic cost).

Concept	**Cost (€)**
Amortization	397,24
Electric cost	4,82
Internet cost	29,3
Travel cost	80
CG cost	**511,36**

Table 20: Generic cost of the project. [Own calculations]

Other Cost

Contingencies

It is very important to be prepared for unforeseen events, which may affect the budget forecast. For this reason, it is necessary to have a contingency fund, which in the IT sector is approximately 10-20

Incidental cost

If a contingency plan were to be activated, the impact on the budget of the work would also have to be calculated. The alternative plans can be found in Section A.2.3. Moreover, it is important to multiply the price it would cost by the risk probability that we need to activate the plan.

Potential risk	**Estimated cost (€)**	**Risk (%)**	**Cost (€)**
Inexperience in the programming language (20 hours)	359	25	89,75
Project deadline (20 hours)	359	30	107,7
Impossibility of meeting with the tutor in person (5 hours)	0	100	0
Serious illness or injury (50 hours)	897,5	5	44,86
Technological failure (5 hours)	89,75	15	13,46
Total			**255,77**

Table 21: Potential risks of the project. [Own calculations]

Total Cost

In Table 22, it is shown the total cost calculated for the project.

Category	**Cost (€)**
CPA cost	14.226,5
CG cost	511,36
Contingency	2.947,57
Incidental cost	255,77
Total	**17.941,2**

Table 22: Total cost of the project. [Own calculations]

Management control

Effective management control is crucial for identifying and correcting any deviations from the initial budget. To achieve this, various monitoring tools and formulas will be implemented to enable a detailed analysis of costs across different project categories, such as personnel, materials, and contingencies. These tools facilitate the adoption of proactive measures whenever a deviation is detected. Following are listed the indicator formulas for the different deviations we can have:

- **[Human Resources] Deviation of hours per task:**
  Deviation of hours per task = (Estimated hours − Actual hours)

- **[Human Resources] Deviation of cost per task:**
  Deviation of cost per task = (Horas estimadas − Horas reales) × Coste real

- **[Contingency] Contingency cost deviation**
  Contingency cost deviation = Estimated cost of contingency−Cost of contingency used

- **[Generic Resources] Amortization deviation:**
  $\textit{Amortization deviation} = (\text{Estimated hours used} - \text{Real hours used}) \times \text{Price per hour}$

In case any of these events happen and adversely affect the budget, we will have to use the budget part reserved for incidents.

A.3.2 Sustainability

Self-assessment

Before the survey I was pretty sure that I had an acceptable understanding of sustainability, when in fact I did not. Answering all the questions has made me realise that the concept of sustainability in a project, and especially as applied to the IT field, is more complex than I previously thought. In particular, I was surprised by the large number of mechanisms and indicators that exist to measure the impact of each decision or action in a project. On the other hand, I always saw sustainability as a big block, when in fact it is key to divide it into 3 blocks: economic, social and environmental in order to go deeper into each aspect. In conclusion, the survey has helped me to realise how little practical knowledge I had about sustainability and how important it is in any project.

Economic Dimension

Have you estimated the cost of undertaking the project (human and material resources)?

The cost of undertaking the project can be found in Section 3.1 of the document. The estimated resources have been divided into several parts: human, hardware, software and material. The methodology used is based on detailed cost estimates, including a rigorous management control to minimise possible budget deviations.

How is the problem that you wish to address resolved currently (state of the art)?

Currently, there are methods that use other similar algorithms (K-Means). My proposal aims to create a new way of approaching the training of RBFNNs which will enrich the field of research and give more possibilities for future research.

In what ways will your solution economically improve existing solutions?

As this is a new proposal, a comparative study with similar previous methods is needed to see if it includes any improvements. In the best case, there will be an improvement in efficiency by improving time and, therefore, costs.

Enviromental Dimension

Have you estimated the environmental impact of undertaking the project?

As this is a project that only requires the use of a computer and no other material, only a study of the electrical energy it will consume has been carried out. In order to draw useful conclusions, the experiments will be repeated several times, which will result in a large expenditure of energy. More details can be found in Section 3.1.2.

Have you considered how to minimise the impact, for example by reusing resources?

As I mentioned before, this project does not use any physical materials that can be reused. The only resource that can be reused is programmed code, which will have no effect on reducing environmental impact.

How is the problem that you wish to address resolved currently (state of the art)?

Currently, other methods are being used which have been tested and are efficient. This study aims to create a new method and experiment with it to see if there are improvements.

In what ways will your solution environmentally improve existing solutions?

Ideally, the study's method would be more efficient than current methods, which would result in less energy consumption and thus less impact on the environment.

Social dimension

What do you think undertaking the project has contributed to you personally?

First, this project will allow me to enter the research field and understand the necessary steps to conduct a study of this nature. Secondly, being the main person responsible for a large project will help me improve my organizational skills and plan projects effectively. Finally, it will give me a great knowledge base on RBFNNs that will be very useful in the sector I want to work in.

How is the problem that you wish to address resolved currently (state of the art)?

Currently, this problem is solved using other basic algorithms commonly accepted by the scientific community. Each algorithm has its pros and cons, so it is important to keep researching and exploring new ways of solving problems.

In what ways will your solution socially improve (quality of life) with respect to other existing solutions?

The aim of this work is to create a new method for training RBFNNs. This could mean that those interested in the field would have more options to apply on RBFNNs, which would bring an enrichment to the field of neural networks.

Is there a real need for the project?

Current methods work well, but in recent fields such as RBFNNs it is always important to investigate and look for new ways of dealing with problems. The field of research is always moving forward.

A.4 Changes and Difficulties

In this section, we will review the initial planning and objectives set out in the final GEP report to see how the work has evolved and explain the different changes that have taken place.

First of all, the objectives have undergone some changes. The main change has been to discard the idea of using GLM for the optimisation of the RBFNN weights. This approach of using GLMs was very tempting, as they are a very powerful tool for dealing with different types of problems and data. The idea was replaced by using only regression data, as the main focus for testing the RBFNN was more on the regularisation part, using different methods and comparing them with each other. Specifically, by focusing more on regularisation, we had to decide on a smaller range of data types to apply regularisation to. Our concern was that by trying to cover a wider range of possibilities, we would not achieve results of sufficient quality and detail.

The change in approach was motivated by the intention to study Orr's theoretical work, which discusses a ridge regression method for regression problems. This has allowed us to create new objectives, thanks to the possibility of comparing Orr's theoretical method with current implementations that use Cross-Validation for regression parameter tuning. In addition, it is possible to compare it with other types of regularisations.

In second place, it was planned to document and create the working memory gradually. This did not make sense from the beginning, because without a strong theoretical basis and a minimum of planning, the hours spent on this task were not worthwhile.

Finally, the programming of the software was mainly divided into creating the hierarchical clustering and then creating the GLM and the regularisation methods. The margin for the theoretical part was too large, as progress was too slow without applying what had been learned in programming. For this reason, it was decided to advance the programming of hierarchical clustering while continuing research in this field.

As far as the methodology is concerned, thanks to good planning, no major changes to the initial proposal were necessary. The use of methods such as Kanban and the division of tasks into sprint-based iterations played a crucial role.

A.5 Knowledge Integration

A.5.1 Programming Knowledge

The programming phase plays a crucial role in this project, as the main goal is to develop a training method for an RBFNN. In particular, it is important to be able to understand code from different sources, be able to construct robust and efficient algorithms and be able to test them. During the process of implementing the method, we have applied knowledge from various subjects, including Programming 1 and 2, EDA, Algorithms and APA. In addition, we have supplemented our existing knowledge with new insights into various Python libraries, such as *scikit-learn* [35].

A.5.2 Statistics and Artificial Intelligence Knowledge

Statistical knowledge plays a key role in developing reliable artificial intelligence (AI) models. In this project, understanding statistical concepts was essential for applying regularisation techniques, such as Ridge, Lasso, and AIC. Statistical metrics, particularly Normalized Mean Squared Error (NMSE), were also essential to evaluate the quality and generalisability of the models.

Furthermore, concepts from the Artificial Intelligence subject and APA allowed us to understand important aspects of Radial Basis Function Neural Networks (RBFNNs). Basic mathematical skills from other studies were also necessary, especially for efficient matrix calculations, Euclidean distance computations, and iterative optimization algorithms.

In short, combining statistical, artificial intelligence, and mathematical knowledge allowed us to thoroughly approach the development and analysis of the machine learning models in this thesis, ensuring their robustness and quality.

A.5.3 Other Knowledge Fields

Knowledge of clustering techniques was essential for this project, as one of the main objectives was to find an efficient way to position neurons in a radial basis function neural network. The key aspect of this area were understanding different linkage criteria, identifying optimal cluster configurations and understanding different clustering techniques.

Some subjects, as Computer Vision, provide a solid base for understanding complex aspects of clustering techniques.

A.6 Laws and Regulations

Although this project is not subject to any specific law or regulation, it relies on third-party software with its own licensing and attribution requirements. As the codebase is written in Python [32], it is covered by the Python Software Foundation License (PSFL). The PSFL allows users to run, share, modify, study and copy Python code without imposing copyleft restrictions, meaning that our use of Python to develop and distribute the RBFNN implementation is fully compliant.

However, once the coding phase is complete, the chosen datasets may carry license terms, so we must check and respect these conditions to avoid any misuse.

References

[1] M. J. L. Orr. *Introduction to Radial Basis Function Networks.* Tech. rep. Centre for Cognitive Science, University of Edinburgh, 1996.

[2] D. S. Broomhead and D. Lowe. "Multivariable Functional Interpolation and Adaptive Networks". *Complex Systems* 2 (1988), pp. 321–355.

[3] J. Moody and C. J. Darken. "Fast learning in networks of locally-tuned processing units". *Neural Computation* 1.2 (1989), pp. 281–294.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning.* New York: Springer, 2006. ISBN: 978-0387310732. DOI: 10.1007/978-0-387-45528-0.

[5] D. Müllner. "Modern Hierarchical, Agglomerative Clustering Algorithms". *arXiv* (2011). arXiv: 1109.2378.

[6] S. Glasmacher. *What is Validation Data Used For? – Machine Learning Basics.* Accessed 2025-06-01. 2022. URL: https://galaxyinferno.com/what-is-validation-data-used-for-machine-learning-basics/.

[7] M. W. Browne. "Cross-Validation Methods". *Journal of Mathematical Psychology* 44.1 (2000), pp. 108–132. DOI: 10.1006/jmps.1999.1279.

[8] A. Arif. *How Cross-Validation Works in Machine Learning.* Accessed 2025-06-11. 2020. URL: https://dataaspirant.com/cross-validation/.

[9] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization". *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[10] U. Tewari. *Regularization — Understanding L1 and L2 Regularization for Deep Learning.* Accessed 2025-05-30. 2021. URL: https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf.

[11] R. Tibshirani. "Regression Shrinkage and Selection via the LASSO". *Journal of the Royal Statistical Society, Series B* 58.1 (1996), pp. 267–288.

[12] A. E. Hoerl and R. W. Kennard. "Ridge Regression: Biased Estimation for Non-Orthogonal Problems". *Technometrics* 12.1 (1970), pp. 55–67. DOI: 10.1080/00401706.1970.10488634.

[13] H. Akaike. "A New Look at the Statistical Model Identification". *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723. DOI: 10.1109/TAC.1974.1100705.

[14] H. Akaike. "Information Theory and an Extension of the Maximum Likelihood Principle". In: *Proceedings of the 2nd International Symposium on Information Theory.* Ed. by B. N. Petrov and F. Csaki. Budapest: Akadémiai Kiadó, 1973, pp. 267–281.

[15] K. Naik. *AIC vs BIC for Regression Models – Formula and Examples.* Accessed 2025-05-29. 2021. URL: https://vitalflux.com/aic-vs-bic-for-regression-models-formula-examples/.

[16] R. Franke. "Scattered data interpolation: tests of some methods". *Mathematics of Computation* 38.157 (1982), pp. 181–200.

[17] C. A. Micchelli. "Interpolation of scattered data: distance matrices and conditionally positive definite functions". *Constructive Approximation* 2.1 (1986), pp. 11–22.

[18] T. Poggio and F. Girosi. "Networks for approximation and learning". *Proceedings of the IEEE* 78.9 (1990), pp. 1481–1497.

[19] J. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability.* Vol. 1. Berkeley: University of California Press, 1967, pp. 281–297.

[20] N. Benoudjit and M. Verleysen. "On the Kernel Widths in Radial-Basis Function Networks". *Neural Processing Letters* 18.2 (2003), pp. 139–154. DOI: 10.1023/A:1026289910256.

[21] E. I. Belhaj. "A Modified Rule-of-Thumb Method for Kernel Density Estimation". *Journal of Mathematical Problems, Equations and Statistics* 5.1 (2024), pp. 143–149. ISSN: 2709-9407.

[22] F. Nielsen. "Hierarchical Clustering". In: *Introduction to HPC with MPI for Data Science.* Springer, 2016. Chap. 8, pp. 195–211. ISBN: 978-3-319-21903-5. DOI: 10.1007/978-3-319-21903-5_8.

[23] F. Cortés. *Tipos de Clustering Jerárquico.* Accessed 2025-06-01. 2022. URL: https://aiplanet.com/notebooks/4381/felipe.cortes/tipos-de-clustering-jerarquico.

[24] Scikit-learn Developers. *Linkage Comparison (Hierarchical clustering) — Scikit-learn Example.* Accessed 2025-05-27. 2023. URL: https://scikit-learn.org/stable/auto_examples/cluster/plot_linkage_comparison.html.

[25] R. Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". *Journal of the Royal Statistical Society, Series C* 22.3 (1973), pp. 238–241. DOI: 10.2307/2346783.

[26] S. C. Johnson. "Hierarchical Clustering Schemes". *Psychometrika* 32.3 (1967), pp. 241–254. DOI: 10.1007/BF02289588.

[27] L. L. McQuitty. "Similarity Analysis by Reciprocal Pairs for Discrete and Continuous Data". *Educational and Psychological Measurement* 26 (1966), pp. 825–831. DOI: 10.1177/001316446602600407.

[28] J. H. Ward. "Hierarchical Grouping to Optimize an Objective Function". *Journal of the American Statistical Association* 58.301 (1963), pp. 236–244. DOI: 10.1080/01621459.1963.10500845.

[29] SciPy Community. *scipy.cluster.hierarchy.linkage.* Accessed 2025-06-02. SciPy. 2024.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In: *Proceedings of the 1996 ACM SIGMOD*

International Conference on Management of Data. ACM, 1996, pp. 103–114. DOI: 10.1145/233269.233324.

[31] S. Guha, R. Rastogi, and K. Shim. “CURE: An Efficient Clustering Algorithm for Large Databases”. In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. ACM, 1998, pp. 73–84. DOI: 10.1145/276304.276312.

[32] P. S. Foundation. *Python 3 Language Reference*. Accessed 2025-06-02. Python Software Foundation. 2024.

[33] Scikit-learn Developers. *sklearn.datasets.make_regression — Generate a Random Regression Problem*. Accessed 2025-06-04. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html.

[34] Scikit-learn Developers. *sklearn.linear_model.RidgeCV — scikit-learn Documentation*. Accessed 2025-06-08. scikit-learn. 2025.

[35] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195.

[36] Scikit-learn Developers. *sklearn.linear_model.LassoCV — scikit-learn Documentation*. Accessed 2025-06-09. scikit-learn. 2024.

[37] A. Miller. *Subset Selection in Regression*. 2nd ed. Boca Raton: Chapman and Hall/CRC, 2002. ISBN: 978-1584881746.

[38] A. Kumar. *Local & Global Minima Explained with Examples*. Accessed 2025-06-02. 2021. URL: https://vitalflux.com/local-global-maxima-minima-explained-examples/.

[39] H. Zou and T. Hastie. “Regularization and Variable Selection via the Elastic Net”. *Journal of the Royal Statistical Society: Series B* 67.2 (2005), pp. 301–320. DOI: 10.1111/j.1467-9868.2005.00503.x.

[40] C. F. Dormann, J. Elith, et al. “Collinearity: A Review of Methods to Deal with It and a Simulation Study Evaluating Their Performance”. *Ecography* 36.1 (2013), pp. 27–46. DOI: 10.1111/j.1600-0587.2012.07348.x.

[41] T. Hastie, R. Tibshirani, and J. Friedman. “Model Assessment and Selection”. In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. New York, NY: Springer, 2009. Chap. 7, pp. 219–259.

[42] G. Casella and R. L. Berger. *Statistical Inference*. 2nd. Pacific Grove, CA: Duxbury, 2002.

[43] D. L. Donoho and I. M. Johnstone. “Ideal Spatial Adaptation by Wavelet Shrinkage”. *Biometrika* 81.3 (1994), pp. 425–455. DOI: 10.1093/biomet/81.3.425.

[44] I.-C. Yeh. *Concrete Compressive Strength*. Accessed 2025-06-10. UCI Machine Learning Repository, 1998. DOI: 10.24432/C5PK67.

[45] I.-C. Yeh. “Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks”. *Cement and Concrete Research* 28.12 (1998), pp. 1797–1808. DOI: `10.1016/S0008-8846(98)00165-3`.

[46] J. Žegklitz and P. Posík. “Learning Linear Feature Space Transformations in Regression Problems Using CMA-ES”. In: *Proceedings of the 23rd European Conference on Genetic Programming (EuroGP)*. Vol. 12101. Lecture Notes in Computer Science. Springer, 2020, pp. 109–125. DOI: `10.1007/978-3-030-44094-7_7`.

[47] A. Shishegaran, H. Varaee, T. Rabczuk, and G. Shishegaran. “High Correlated Variables Creator Machine: Prediction of the Compressive Strength of Concrete”. *arXiv* (2020). arXiv: `2009.06421 [cs.LG]`.

[48] P. S. Rana. *Physicochemical Properties of Protein Tertiary Structure*. UCI Machine Learning Repository. 2013. DOI: `10.24432/C5QW3H`.