Appendix

K-Nearest Neighbour (kNN)

The kNN method is a straightforward machine learning algorithm commonly employed for both classification and regression tasks. When applied to regression, the kNN algorithm operates by assigning the property value of an object as the average of the values belonging to its k nearest neighbors (Kramer 2013).

In kNN implementations, two essential parameters need to be determined: the number of neighbors (*k*) and the type of distance metric used to determine proximity between instances. Selecting an appropriate value for *k* is crucial as it directly impacts the smoothness and accuracy of the predictions. Additionally, the choice of distance measure is important as it determines how the algorithm quantifies the similarity or dissimilarity between data points.

The problem in regression is to predict labels $\hat{y} \in \mathbb{R}^d$ for new patterns $\hat{x} \in \mathbb{R}^q$ based on a set of N observations, i.e., labeled patterns $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$. In kNN regression, $\hat{y}$ is estimated as the mean value of the k-nearest neighbors:

$$\hat{y} = \frac{1}{N} \sum_{i \in N_k(\hat{x})} y_i, \tag{8}$$

where $N_k(\hat{x})$ contains the indices of the k-nearest neighbors of $\hat{x}$. Several distance metrics are commonly used to measure the similarity or dissimilarity between data points in kNN, some of which are Euclidean, Mahalanobis, Manhattan, Minkowski, Chebyshev, Cosine, Correlation, Hamming, and Jaccard, among others (Prasath et al. 2019).

The kNN algorithm produces locally constant outputs, leading to the formation of plateaus within the output space. However, from an optimization perspective, these plateaus can hinder the efficiency of optimization methods in effectively approximating the optimal solution. The existence of plateaus limits the available information for identifying promising search directions during the optimization process, thereby impeding the rapid convergence of optimization methods toward the optimal solution. To address this issue, Bailey (Bailey 1978) introduced the distance-weighted kNN approach, which assigns weights to each neighbor's contribution in the estimation based on its distance to $\hat{x}$:

$$\hat{y} = \sum_{i \in N_k(\hat{x})} \frac{w(x_i)}{\sum_{j \in N_k(\hat{x})} w(x_j)} y_i, \tag{9}$$

with:

$$w(x_i) = \frac{1}{\|\hat{x} - x_i\|^2}, \tag{10}$$

Hence, closer neighbors have a higher impact on the prediction.

Random Forest Regression (RFR)

Random Forest Regression (RFR) is a powerful and versatile supervised learning algorithm for regression tasks. It belongs to the family of ensemble methods, combining the principles of

decision trees and bagging. RFR is particularly effective when dealing with complex datasets, as it can handle numerical and categorical variables.

In RFR, an ensemble of decision trees is built to create a robust predictive model. Each decision tree is trained on a randomly selected subset of the training data and a random subset of features, which helps reduce overfitting and increase model accuracy. During training, the decision trees learn to make predictions by partitioning the feature space into smaller regions based on the values of the input features.

To make predictions with RFR, the algorithm combines the predictions of all individual decision trees in the ensemble, as illustrated in Figure 15. The final prediction is determined by averaging or taking the median of the predictions from the individual trees, depending on the specific regression problem. This aggregation process helps to reduce the impact of outliers and noise, leading to more stable and accurate predictions.

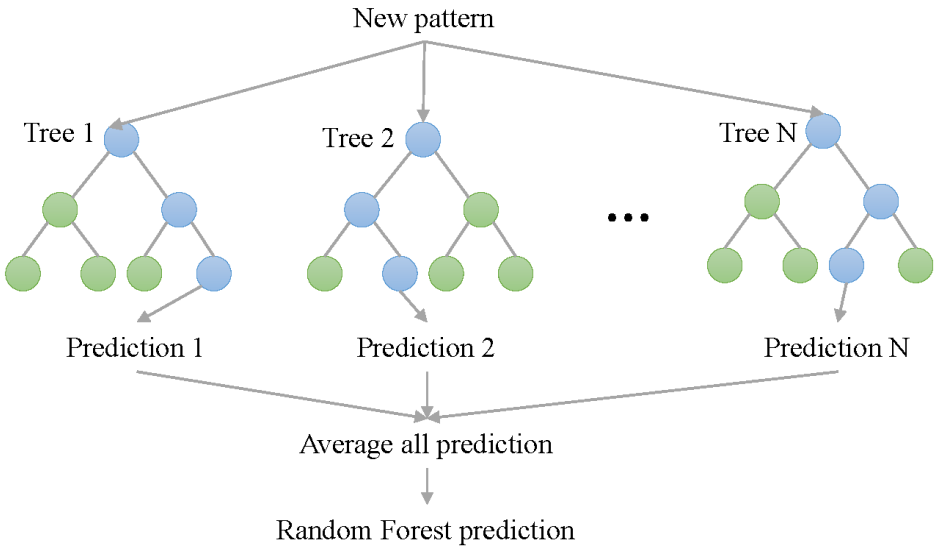

Figure 15. Scheme of Random Forest.

Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) are machine learning models inspired by the structure and function of biological neural networks (Hagan et al. 2014). As shown in Figure 16, ANNs consist of an input layer, an output layer, and one or more hidden layers in between. Each neuron in the input layer receives an input signal transmitted through the network via weighted connections between neurons. The neurons in each hidden layer perform a non-linear transformation on the input signal, and the output of each neuron is determined by an activation function, such as a sigmoid or ReLU function. The output layer produces the final output of the network.

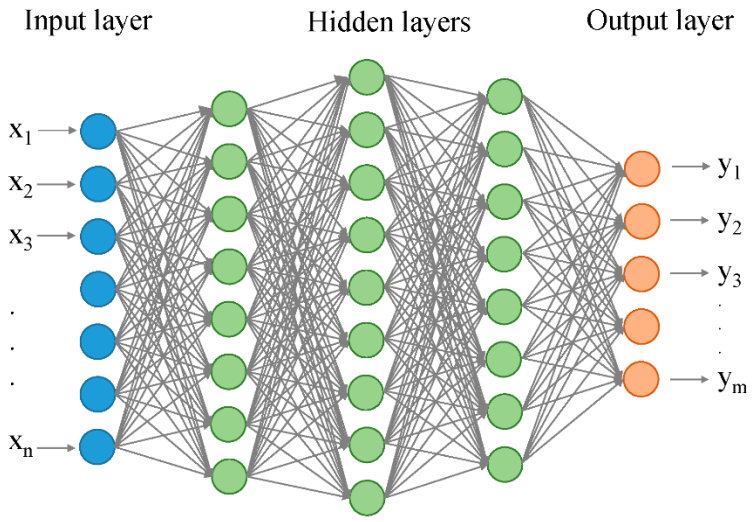

Figure 16. Scheme of an ANN.

One of the key features of ANNs is that they are feedforward networks, meaning that information flows in only one direction, from the input layer to the output layer. This prevents cycles or loops in the network and ensures that the network's output is well-defined for a given input. As an example, the outputs of a three-layer ANN are given by

$$y_i = f\left(\sum_j w_{ij} f\left(\sum_k v_{jk} x_k + b_{vk}\right) + b_{wi}\right) \tag{11}$$

where $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ and $\boldsymbol{y} = \{y_1, y_2, \dots, y_m\}$ are the input and output vectors, $\mathbf{w}$ and $\mathbf{v}$ are the interconnection weights, $\mathbf{b}$ represents the bias (or threshold) terms and f is the transfer function, usually a sigmoid function. The training of an ANN involves adjusting the weights and biases of the connections between neurons so that the network produces accurate outputs for a given set of inputs. This is typically achieved through a process called backpropagation, which involves calculating the error between the network's output and the desired output, and then propagating this error back through the network to adjust the weights and biases. Training aims to minimize the difference between the network's output and the desired output, usually measured using a loss function such as mean squared error. Once trained, the network can make predictions on new, unseen data.

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a heuristic optimization technique inspired by the social behavior of birds flocking or fish schooling. Introduced by Eberhart and Kennedy in 1995 (Eberhart and Kennedy 1995), this algorithm simulates the collective behavior of a swarm to search for optimal solutions in a multidimensional space.

A 'swarm' in PSO comprises potential solutions to a given problem, each termed as a 'particle'. Each particle has a position representing a potential solution and a velocity dictating the change in position over iterations. The particles 'fly' through the solution space by adjusting

their positions based on their own best-known position and the global best-known position of the swarm.

The particles are randomly initialized with positions and velocities. Then, the fitness of each particle (i.e., the quality of the solution it represents) is evaluated using a predefined objective function. Each particle updates its velocity and position based on a combination of its historical best position, the swarm's best position, and some stochastic elements. Specifically, the new velocity and position are updated using the formula:

$$v_i = wv_i + c_1 r_1 (p_{best} - x_i) + c_2 r_2 (g_{best} - x_i) \quad (12)$$
$$x_i = x_i + v_i$$

Where v_i denotes the *i*th particle's velocity, while *w* acts as an inertia weight that moderates its momentum. The coefficients c_1 and c_2 represent cognitive and social scaling factors, respectively. Both r_1 and r_2 are random values selected from the interval [0,1]. The terms $p_{best,i}$ and g_{best} refer to the *i*th particle's optimal historical position and the swarm's overall optimal position, respectively. The symbol x_i indicates the *i*th particle's current position. The PSO process concludes either upon reaching a predetermined maximum number of iterations or when specific conditions are satisfied, such as a negligible variation in g_{best} over consecutive iterations.

Due to its simplicity, flexibility, and robustness, PSO has been applied to a wide variety of optimization problems, including function optimization, neural network training, clustering, and many others (Poli, Kennedy, and Blackwell 2007).