CHAPTER IV

RESULTS

This chapter presents the theoretical and algorithmic contributions of this research, establishing the mathematical foundations for Double Generalized Additive Models (DGAMs) enhanced through integration with advanced machine learning techniques. The chapter systematically demonstrates the theoretical feasibility, algorithmic specifications, and mathematical rigor underlying the proposed methodologies.

4.1 Overview and Structure

The primary objective of this chapter is to provide rigorous mathematical proofs and algorithmic formulations that establish the theoretical validity of extending classical Generalized Additive Models to simultaneously model both mean and dispersion functions using five distinct machine learning techniques: Smoothing Splines, Explainable Boosting Machines (EBM), Extreme Gradient Boosting (XGBoost), Random Forests (RF), and Support Vector Regression (SVR).

This chapter addresses the first major research objective articulated in Phase I of the methodology: *to develop theoretical concepts and computational algorithms for flexible generalized additive models of the mean and dispersion functions*. The mathematical developments presented here establish the theoretical foundation necessary for subsequent computational implementation and empirical validation.

4.1.1 Structure of This Chapter

The chapter is structured as follows:

Section 4.1: Conceptual Framework We begin by formally defining the Double Generalized Additive Model framework, establishing the mathematical foundations that extend

classical GAMs to accommodate flexible dispersion modeling. This section provides the formal specification of the DGAM structure for Gaussian responses, probabilistic foundations from the exponential dispersion family, dual link function formulation for mean and dispersion, and mathematical conditions ensuring model identifiability and estimability.

Sections 4.2–4.7: Technique-Specific Developments For each of the five machine learning techniques, we provide:

- **Mathematical Formulation** with rigorous derivation of how each technique is adapted to the DGAM framework, including objective functions, penalization mechanisms, and coupling between mean and dispersion estimation;
- **Theoretical Properties** with formal proofs establishing existence and uniqueness of solutions, consistency and asymptotic normality of estimators, convergence guarantees, and identifiability conditions; and
- **Algorithmic Specification** with detailed computational algorithms, alternating optimization strategies, convergence criteria, and complexity analysis.

Each technique-specific section demonstrates the *mathematical feasibility* of integrating that particular method with the DGAM framework, providing the theoretical justification necessary for subsequent computational implementation.

4.1.2 Purpose and Contribution

The mathematical and algorithmic developments in this chapter serve three critical purposes:

1. Theoretical Validation : The formal mathematical proofs establish that the proposed DGAM extensions are theoretically sound, with well-defined optimization objectives, identifiable parameters, and desirable asymptotic properties.

2. Algorithmic Specification : The detailed algorithms provide precise computational recipes that can be directly translated into executable code, bridging the gap between abstract mathematical theory and concrete computational implementation.

3. Methodological Innovation : By systematically demonstrating how five distinct machine learning techniques can be adapted to the DGAM framework, this chapter establishes a general methodological template for extending flexible dispersion modeling to other advanced methods.

4.1.3 Relationship to Empirical Validation

The mathematical proofs and algorithms presented in this chapter establish *what is theoretically possible*—demonstrating that DGAMs with flexible machine learning components are mathematically well-defined and computationally tractable. However, theoretical feasibility alone does not guarantee practical utility.

The computational implementations based on these algorithms, along with their empirical evaluation on real-world health data, are presented in the Appendix (Appendix A), which contains complete Python code implementations of all algorithms, detailed experimental results across 16 prediction tasks, comprehensive performance comparisons between GAM and DGAM frameworks, statistical significance tests and model selection analyses, and visualization of fitted mean and dispersion functions.

This organizational structure—theoretical development in the main chapter, computational implementation in the Appendix—reflects the logical progression from mathematical possibility to practical realization.

4.1.4 Key Contributions

The primary contributions of this chapter include:

1. **Mathematical Framework:** Rigorous formalization of DGAMs for Gaussian responses with identity link for mean and log link for dispersion.
2. **Algorithmic Integration:** Development of five distinct computational algorithms (Algorithms 1–5) that integrate Smoothing Splines, EBM, XGBoost, RF, and SVR with the DGAM framework.
3. **Theoretical Properties:** Formal proofs of consistency, convergence, and identifiability for DGAM estimators under each technique.

4. **Coupled Estimation Procedures:** Novel coupled Penalized Iteratively Reweighted Least Squares (PIRLS) and alternating boosting/bagging algorithms that properly account for the interdependence between mean and dispersion parameters.

5. **Computational Specifications:** Complete algorithmic specifications with precise stopping criteria, hyperparameter settings, and complexity analysis.

4.1.5 Reading Guide

Readers primarily interested in theoretical foundations should focus on the mathematical formulations and theoretical property proofs within each technique-specific section. Those interested in computational implementation should pay particular attention to the algorithmic specifications and detailed algorithm steps, which can be directly translated into code. Readers seeking empirical evidence of model performance should proceed directly to the Appendix (Appendix **??**), where the Python implementations are applied to real-world health datasets.

The remainder of this chapter presents the detailed mathematical and algorithmic developments for DGAMs, beginning with the foundational framework definition and proceeding through each of the five machine learning techniques.

4.2 Double Generalized Additive Models: Mathematical Framework

A Double Generalized Additive Model (DGAM) extends Generalized Additive Models (GAMs) by simultaneously modeling both the mean and dispersion (variance) of the response variable as smooth functions of covariates. While traditional GAMs focus solely on modeling the mean response, DGAMs recognize that variability in data may also change systematically with predictor variables, making them particularly valuable for analyzing heteroscedastic data where variance is not constant (Rigby and Stasinopoulos, 2005; Wood, 2017).

DGAMs are especially useful in healthcare, environmental science, and finance, where both the central tendency and the spread of outcomes may depend on explanatory variables. For instance, in medical studies, not only might the average health out-

come vary with age and treatment, but the variability in individual responses may also change across different patient subgroups.

The Double Generalized Additive Model specifies separate additive predictor structures for both location (mean) and scale (dispersion) parameters. Instead of assuming a constant dispersion parameter ϕ as in GLMs and GAMs, DGAMs allow ϕ_i to vary across observations as a function of covariates.

4.2.1 Random Component

The distribution of the response variable Y_i for each of the n independently sampled observations follows a distribution from the exponential dispersion family:

$$f_{y_i}(y_i; \theta_i, \phi_i) = \exp\left(\frac{y_i\theta_i - b(\theta_i)}{a(\phi_i)} + c(y_i; \phi_i)\right),$$

where $\theta_i \in \Theta$ is the canonical parameter, $\phi_i > 0$ is the observation-specific dispersion parameter, $a(\cdot)$ is a known function (typically $a(\phi_i) = \phi_i$), $b(\cdot)$ is the cumulant function, and $c(\cdot)$ is a normalization constant. Common distributions include Gaussian, Gamma, and Inverse Gaussian.

As in GLMs and GAMs, the mean and variance relationships are:

$$\mathbb{E}[Y_i] = b'(\theta_i) = \mu_i,$$
$$\text{Var}[Y_i] = a(\phi_i)b''(\theta_i) = \phi_i V(\mu_i),$$

where $V(\mu_i)$ is the variance function depending on the chosen distribution family.

4.2.2 Mean Structure (Location Model)

The mean model replaces the linear predictor with an additive predictor composed of smooth functions:

$$g(\mu_i) = \eta_i = A_i\boldsymbol{\theta} + f_1(x_{1i}) + f_2(x_{2i}) + \ldots + f_p(x_{pi}),$$

where $g(\cdot)$ is a monotonic link function (e.g., identity, log, logit), A_i is a $1 \times r$

row vector of covariates for strictly parametric terms, $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_r)^T$ is the $r \times 1$ vector of parametric coefficients (ensuring $A_i\boldsymbol{\theta}$ is a scalar), $f_j(\cdot)$ are smooth, potentially nonlinear functions of covariates x_{ji} for $j = 1, 2, \ldots, p$, and η_i is the linear predictor for the mean structure.

The smooth functions f_j are typically estimated using penalized splines, smoothing splines, or other basis function representations.

4.2.3 Dispersion Structure (Scale Model)

A key innovation of DGAMs is the explicit modeling of the dispersion parameter as a function of covariates:

$$h(\phi_i) = \lambda_i = B_i\boldsymbol{\gamma} + s_1(z_{1i}) + s_2(z_{2i}) + \ldots + s_q(z_{qi}),$$

where $h(\cdot)$ is a monotonic link function for the dispersion (commonly log to ensure $\phi_i > 0$), B_i is a row vector of covariates for parametric terms in the dispersion model, $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_m)^T$ is the vector of parametric coefficients for dispersion, $s_j(\cdot)$ are smooth functions of covariates z_{ji} for $j = 1, 2, \ldots, q$, and λ_i is the linear predictor for the dispersion structure.

The covariates in the dispersion model may be the same as or different from those in the mean model, allowing investigation of whether certain variables affect only the mean, only the variability, or both.

4.2.4 Link Functions

DGAMs employ two separate smooth and invertible link functions to connect the parameters of interest to their respective additive predictors.

Mean Link Function for Gaussian Response The function $g(\cdot)$ relates the mean μ_i to the additive predictor η_i:

$$g(\mu_i) = \eta_i = A_i\boldsymbol{\theta} + f_1(x_{1i}) + f_2(x_{2i}) + \ldots + f_p(x_{pi}).$$

For Gaussian (normal) distributed response variables, the canonical link function is the identity link:

Link function name	Link function formula	Inverse link
Identity	$g(\mu) = \mu$	$g^{-1}(\eta) = \eta$

The identity link is natural for Gaussian responses as it imposes no restrictions on the range of μ_i, which is appropriate since the mean of a normal distribution can take any real value.

Dispersion Link Function for Gaussian Response The function $h(\cdot)$ relates the dispersion parameter ϕ_i (representing the variance σ_i^2) to the additive predictor λ_i:

$$h(\phi_i) = \lambda_i = B_i\boldsymbol{\gamma} + s_1(z_{1i}) + s_2(z_{2i}) + \ldots + s_q(z_{qi}).$$

For the dispersion parameter, which must be strictly positive ($\phi_i > 0$), the most commonly used link function is:

Link function name	Link function formula	Inverse link
Log	$h(\phi) = \log(\phi)$	$h^{-1}(\lambda) = \exp(\lambda)$

The log link naturally ensures $\phi_i > 0$ for all values of λ_i.

Inverse Link Functions The inverse link functions $g^{-1}(\cdot)$ and $h^{-1}(\cdot)$ transform the fitted additive predictors back to the original scale:

$$\mu_i = g^{-1}(\eta_i) = g^{-1}\left(A_i\boldsymbol{\theta} + \sum_{j=1}^{p} f_j(x_{ji})\right) = A_i\boldsymbol{\theta} + \sum_{j=1}^{p} f_j(x_{ji}),$$
$$\phi_i = h^{-1}(\lambda_i) = h^{-1}\left(B_i\boldsymbol{\gamma} + \sum_{j=1}^{q} s_j(z_{ji})\right) = \exp\left(B_i\boldsymbol{\gamma} + \sum_{j=1}^{q} s_j(z_{ji})\right),$$

for $i = 1, 2, \ldots, n$.

4.2.5 Model Specification Summary

The complete Gaussian DGAM specification can be summarized as:

$$g(\mu_i) = \mu_i = A_i\boldsymbol{\theta} + \sum_{j=1}^{p} f_j(x_{ji}),$$
$$h(\phi_i) = \log(\phi_i) = B_i\boldsymbol{\gamma} + \sum_{j=1}^{q} s_j(z_{ji}),$$

with $Y_i \sim \mathcal{N}(\mu_i, \phi_i)$ for $i = 1, 2, \ldots, n$, where $\phi_i = \sigma_i^2$ represents the variance parameter.

Equivalently, using inverse link functions:

$$\mu_i = A_i\boldsymbol{\theta} + \sum_{j=1}^{p} f_j(x_{ji}),$$
$$\sigma_i^2 = \exp\left(B_i\boldsymbol{\gamma} + \sum_{j=1}^{q} s_j(z_{ji})\right).$$

This dual-structure framework enables Gaussian DGAMs to provide comprehensive characterization of how both central tendency (mean) and variability (variance) in the response variable change with predictor variables, making them particularly powerful for modeling heteroscedastic data.

4.3 Smoothing Splines

Smoothing splines provide a flexible nonparametric approach to estimating smooth functions by balancing data fit and smoothness through penalized optimization. Given observations $(y_1, x_1), \ldots, (y_n, x_n)$ and a choice of $\lambda \geq 0$, the smoothing spline estimator $\hat{f}_\lambda$ is defined by:

$$\hat{f}_\lambda = \underset{f \in C^2}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int [f''(t)]^2 dt,$$

where C^2 denotes the space of twice-differentiable functions. The solution is a natural cubic spline with knots at the observed x values, making this optimization problem finite-dimensional (Chouldechova and Hastie, 2015).

4.3.1 Basis Representation and Matrix Formulation

In practice, the smooth function $f(x)$ is represented using a basis expansion. Let $\{b_1(x), b_2(x), \ldots, b_M(x)\}$ denote a set of basis functions (e.g., B-splines or natural cubic splines):

$$f(x) = \sum_{m=1}^{M} b_m(x)\beta_m = \mathbf{b}(x)^T\boldsymbol{\beta},$$

where $\mathbf{b}(x) = (b_1(x), b_2(x), \ldots, b_M(x))^T$ is the basis function vector and $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_M)^T$ are the coefficients to be estimated.

Let B denote the $n \times M$ design matrix with elements $B_{im} = b_m(x_i)$. The fitted values are $\hat{\mathbf{y}} = B\boldsymbol{\beta}$. The roughness penalty can be expressed as:

$$\int [f''(t)]^2 dt = \boldsymbol{\beta}^T S \boldsymbol{\beta},$$

where S is the $M \times M$ penalty matrix with elements $S_{mn} = \int b_m''(x) b_n''(x) dx$.

The penalized least squares objective becomes:

$$\hat{\boldsymbol{\beta}}_\lambda = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\mathbf{y} - B\boldsymbol{\beta}\|^2 + \lambda \boldsymbol{\beta}^T S \boldsymbol{\beta},$$

with closed-form solution:

$$\hat{\boldsymbol{\beta}}_\lambda = (B^T B + \lambda S)^{-1} B^T \mathbf{y}.$$

4.3.2 Parameter Estimation for DGAM with Smoothing Splines

For DGAMs, smoothing splines represent both mean and dispersion functions:

$$\eta_{\mu,i} = (B_\mu)_i \boldsymbol{\beta}_\mu = \sum_{m=1}^{M_\mu} b_{\mu,m}(x_i)\beta_{\mu,m},$$
$$\eta_{\phi,i} = (B_\phi)_i \boldsymbol{\beta}_\phi = \sum_{m=1}^{M_\phi} b_{\phi,m}(x_i)\beta_{\phi,m},$$

where B_μ is the $n \times M_\mu$ design matrix for the mean model, B_ϕ is the $n \times M_\phi$ design matrix for the dispersion model, $\boldsymbol{\beta}_\mu$ and $\boldsymbol{\beta}_\phi$ are the coefficient vectors.

The mean and dispersion are obtained via inverse link functions:

$$\mu_i = g_\mu^{-1}(\eta_{\mu,i}) = g_\mu^{-1}((B_\mu)_i \boldsymbol{\beta}_\mu),$$
$$\phi_i = g_\phi^{-1}(\eta_{\phi,i}) = g_\phi^{-1}((B_\phi)_i \boldsymbol{\beta}_\phi).$$

The penalized log-likelihood for the DGAM with smoothing splines is:

$$\ell_p(\boldsymbol{\beta}_\mu, \boldsymbol{\beta}_\phi) = \sum_{i=1}^{n} \log p(y_i \mid \mu_i, \phi_i) - \frac{1}{2}\lambda_\mu \boldsymbol{\beta}_\mu^T S_\mu \boldsymbol{\beta}_\mu - \frac{1}{2}\lambda_\phi \boldsymbol{\beta}_\phi^T S_\phi \boldsymbol{\beta}_\phi,$$

where $p(y_i \mid \mu_i, \phi_i)$ is the probability density, S_μ and S_ϕ are the penalty matrices, and λ_μ and λ_ϕ are smoothing parameters.

4.3.3 Algorithm: Classical Spline DGAM via Coupled PIRLS

The estimation proceeds via coupled Penalized Iteratively Reweighted Least Squares (PIRLS):

Algorithm 1 Classical Spline DGAM via Coupled PIRLS

Input: $\{(y_i, x_i)\}_{i=1}^n$, link functions g_μ, g_ϕ, distribution $p(y \mid \mu, \phi)$

Hyperparameters: Spline bases B_μ, B_ϕ, penalties S_μ, S_ϕ, smoothing $\lambda_\mu, \lambda_\phi$

1: **Trainable parameters:** $\boldsymbol{\beta}_\mu, \boldsymbol{\beta}_\phi$
2: Construct design matrices $X_\mu = B_\mu(X)$, $X_\phi = B_\phi(X)$
3: Initialize $\boldsymbol{\beta}_\mu^{(0)}, \boldsymbol{\beta}_\phi^{(0)}$
4: **repeat**
5: Compute $\boldsymbol{\eta}_\mu = X_\mu \boldsymbol{\beta}_\mu$, $\boldsymbol{\eta}_\phi = X_\phi \boldsymbol{\beta}_\phi$
6: Update $\boldsymbol{\mu} = g_\mu^{-1}(\boldsymbol{\eta}_\mu)$, $\boldsymbol{\phi} = g_\phi^{-1}(\boldsymbol{\eta}_\phi)$
7: Evaluate penalized log-likelihood $\ell_p(\boldsymbol{\beta}_\mu, \boldsymbol{\beta}_\phi)$
8: Compute coupled score vectors U_μ, U_ϕ
9: Compute observed information blocks $I_{\mu\mu}, I_{\mu\phi}, I_{\phi\phi}$
10: Solve joint Newton update for $(\boldsymbol{\beta}_\mu, \boldsymbol{\beta}_\phi)$:
11: $$\begin{bmatrix} I_{\mu\mu} & I_{\mu\phi} \\ I_{\mu\phi}^T & I_{\phi\phi} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\beta}_\mu \\ \Delta\boldsymbol{\beta}_\phi \end{bmatrix} = \begin{bmatrix} U_\mu \\ U_\phi \end{bmatrix}$$
12: Update $\boldsymbol{\beta}_\mu \leftarrow \boldsymbol{\beta}_\mu + \Delta\boldsymbol{\beta}_\mu$, $\boldsymbol{\beta}_\phi \leftarrow \boldsymbol{\beta}_\phi + \Delta\boldsymbol{\beta}_\phi$
13: **until** convergence: $\|\Delta\boldsymbol{\beta}_\mu\| < \epsilon_\mu$ and $\|\Delta\boldsymbol{\beta}_\phi\| < \epsilon_\phi$
14: **Output:** $\hat{\mu}(x)$ and $\hat{\phi}(x)$

Detailed Algorithm Steps:

Step 1: Initialization (Line 3) Initialize $\boldsymbol{\beta}_\mu^{(0)}$ and $\boldsymbol{\beta}_\phi^{(0)}$ using GLM estimates with constant dispersion.

Step 2: Compute Score Vectors (Lines 8-9) At iteration t, compute:

$$U_\mu = B_\mu^T W_\mu (\mathbf{y} - \boldsymbol{\mu}) - \lambda_\mu S_\mu \boldsymbol{\beta}_\mu,$$
$$U_\phi = B_\phi^T W_\phi \mathbf{d} - \lambda_\phi S_\phi \boldsymbol{\beta}_\phi,$$

where W_μ and W_ϕ are weight matrices, and $\mathbf{d}$ is the vector of adjusted responses for dispersion.

Step 3: Compute Information Matrix Blocks (Line 9)

$$I_{\mu\mu} = B_\mu^T W_\mu B_\mu + \lambda_\mu S_\mu,$$
$$I_{\phi\phi} = B_\phi^T W_\phi B_\phi + \lambda_\phi S_\phi,$$
$$I_{\mu\phi} = B_\mu^T W_{\mu\phi} B_\phi,$$

where $W_{\mu\phi}$ captures coupling between mean and dispersion.

Step 4: Final Predictions (Line 13) Compute fitted functions:

$$\hat{\mu}(x) = g_\mu^{-1}(\mathbf{b}_\mu(x)^T \hat{\boldsymbol{\beta}}_\mu),$$
$$\hat{\phi}(x) = g_\phi^{-1}(\mathbf{b}_\phi(x)^T \hat{\boldsymbol{\beta}}_\phi).$$

This coupled PIRLS algorithm ensures proper accounting of interdependence between mean and dispersion during estimation.

4.4 Explainable Boosting Machine

The Explainable Boosting Machine (EBM) is an interpretable machine learning algorithm within the InterpretML framework that combines the transparency of Generalized Additive Models with modern boosting techniques (Nori et al., 2019; Lou et al., 2012). EBM maintains interpretability while achieving prediction accuracy comparable to sophisticated ensemble methods.

4.4.1 EBM Model Structure

EBM adopts a Generalized Additive Model structure:

$$g(\mathbb{E}[Y]) = \beta_0 + \sum_{j=1}^{p} f_j(x_j),$$

where $g(\cdot)$ is a link function, β_0 is an intercept term, and $f_j(\cdot)$ are univariate shape functions capturing each feature's contribution.

For models requiring interaction terms:

$$g(\mathbb{E}[Y]) = \beta_0 + \sum_{j=1}^{p} f_j(x_j) + \sum_{i<j} f_{ij}(x_i, x_j),$$

where $f_{ij}(x_i, x_j)$ represents pairwise interactions.

4.4.2 Key Features of EBM

Modern Learning Techniques: Unlike traditional GAMs using splines, EBM employs gradient boosting to learn each feature function f_j with carefully constrained training on one feature at a time in round-robin fashion with very low learning rate.

Round-Robin Feature Selection: By cycling through features sequentially rather than selecting features based on gradient magnitude, EBM learns optimal feature functions independently, revealing unbiased feature contributions.

Automatic Interaction Detection: EBM can automatically detect and incorporate significant pairwise interaction terms, capturing non-additive effects while maintaining interpretability.

4.4.3 Parameter Estimation for DGAM with EBM

For DGAMs, EBM is applied separately to model both mean and dispersion:

$$\eta_{\mu,i} = f_\mu(x_i) = \beta_{0,\mu} + \sum_{j=1}^{p} f_{\mu,j}(x_{ji}),$$
$$\eta_{\phi,i} = f_\phi(x_i) = \beta_{0,\phi} + \sum_{j=1}^{q} f_{\phi,j}(z_{ji}),$$

where $f_\mu(\cdot)$ and $f_\phi(\cdot)$ are additive predictors learned through boosting.

The mean and dispersion parameters are obtained via inverse link functions:

$$\mu_i = g_\mu^{-1}(\eta_{\mu,i}) = g_\mu^{-1}(f_\mu(x_i)),$$
$$\phi_i = g_\phi^{-1}(\eta_{\phi,i}) = g_\phi^{-1}(f_\phi(x_i)).$$

4.4.4 Algorithm: EBM-DGAM via Alternating Boosting

The estimation proceeds through alternating boosting on coupled negative log-likelihood gradients:

Algorithm 2 EBM-DGAM via Alternating Boosting for Mean and Dispersion

Input: $\{(y_i, x_i)\}_{i=1}^n$, link functions g_μ, g_ϕ, distribution $p(y \mid \mu, \phi)$
Hyperparameters: Boosting rounds T, learning rates ν_μ, ν_ϕ, EBM constraints (bins/interactions), early stopping tolerance ε
Trainable parameters: Additive predictors $f_\mu(\cdot), f_\phi(\cdot)$ with $\eta_\mu = f_\mu(x)$, $\eta_\phi = f_\phi(x)$
Initialize $f_\mu^{(0)}(x) \equiv c_\mu$ and $f_\phi^{(0)}(x) \equiv c_\phi$
for $t = 0, 1, \ldots, T-1$ **do**
  Compute linear predictors $\eta_{\mu,i}^{(t)} = f_\mu^{(t)}(x_i)$ and $\eta_{\phi,i}^{(t)} = f_\phi^{(t)}(x_i)$
  Update distributional parameters $\mu_i^{(t)} = g_\mu^{-1}(\eta_{\mu,i}^{(t)})$, $\phi_i^{(t)} = g_\phi^{-1}(\eta_{\phi,i}^{(t)})$
  Evaluate negative log-likelihood $L^{(t)} = \sum_{i=1}^n -\log p(y_i \mid \mu_i^{(t)}, \phi_i^{(t)})$
  Compute mean pseudo-residuals:
    $r_{\mu,i}^{(t)} = -\frac{\partial}{\partial \eta_{\mu,i}} \left[-\log p(y_i \mid \mu_i^{(t)}, \phi_i^{(t)}) \right]$
  Fit EBM base-learner $h_\mu^{(t)}$ to $\{(x_i, r_{\mu,i}^{(t)})\}_{i=1}^n$
  Update $f_\mu^{(t+1)} \leftarrow f_\mu^{(t)} + \nu_\mu h_\mu^{(t)}$
  Recompute $\mu_i^{(t+1)}$ using $f_\mu^{(t+1)}$ while holding $f_\phi^{(t)}$ fixed
  Compute dispersion pseudo-residuals:
    $r_{\phi,i}^{(t)} = -\frac{\partial}{\partial \eta_{\phi,i}} \left[-\log p(y_i \mid \mu_i^{(t+1)}, \phi_i^{(t)}) \right]$
  Fit EBM base-learner $h_\phi^{(t)}$ to $\{(x_i, r_{\phi,i}^{(t)})\}_{i=1}^n$
  Update $f_\phi^{(t+1)} \leftarrow f_\phi^{(t)} + \nu_\phi h_\phi^{(t)}$
  if early stopping criterion satisfied (e.g., $|L^{(t+1)} - L^{(t)}| < \varepsilon$) **then**
    break
  end if
end for
Output: $\hat{\mu}(x) = g_\mu^{-1}(f_\mu(x))$ and $\hat{\phi}(x) = g_\phi^{-1}(f_\phi(x))$

Detailed Algorithm Steps:

Step 1: Initialization (Line 4) Initialize mean and dispersion functions as constants: $f_\mu^{(0)}(x) = c_\mu$ and $f_\phi^{(0)}(x) = c_\phi$, typically based on overall mean and variance of the response.

Step 2: Mean Boosting (Lines 9-12) Compute the gradient of negative log-likelihood with respect to mean linear predictor. For Gaussian distribution with identity link for mean and log link for dispersion:

$$r_{\mu,i}^{(t)} = \frac{y_i - \mu_i^{(t)}}{\phi_i^{(t)}}.$$

Fit EBM base-learner $h_\mu^{(t)}$ to pseudo-residuals and update mean function with learning rate ν_μ.

Step 3: Dispersion Boosting (Lines 13-16) After updating mean, compute gradient with respect to dispersion. For Gaussian distribution with log link:

$$r_{\phi,i}^{(t)} = \frac{(y_i - \mu_i^{(t+1)})^2}{\phi_i^{(t)}} - 1.$$

Fit EBM base-learner $h_\phi^{(t)}$ to dispersion pseudo-residuals and update with learning rate ν_ϕ.

Step 4: Convergence Check (Lines 17-18) Monitor change in negative log-likelihood. If $|L^{(t+1)} - L^{(t)}| < \varepsilon$, terminate to prevent overfitting.

This alternating boosting approach allows EBM to learn complex mean and dispersion functions while maintaining interpretability through additive structure and enabling detailed examination of individual feature contributions through partial dependence plots.

4.5 XGBoost

Extreme Gradient Boosting (XGBoost) is a scalable and efficient machine learning algorithm that combines weak learners, typically regression trees, to create a powerful predictive ensemble model (Chen and Guestrin, 2016). XGBoost employs gradient boosting, a sequential ensemble technique where new trees are built to model the residuals (prediction errors) from previous trees, progressively focusing on the unexplained variance in the training set.

4.5.1 XGBoost Model Structure

The XGBoost algorithm constructs an additive ensemble of regression trees. For a dataset with n observations, the prediction is given by:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i),$$

where K is the number of trees, f_k represents the k-th tree function, and $\phi(x_i)$ is the ensemble prediction for observation i.

At the beginning of the algorithm, each observation is predicted using a constant initialization (typically the mean of the response variable). The algorithm then sequentially adds trees, with each new tree fit to approximate the negative gradient (first derivative) and curvature (second derivative) of the loss function with respect to current predictions.

4.5.2 Objective Function

The XGBoost training objective combines a loss function with a regularization term to control model complexity:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k),$$

where:

- $l(\hat{y}_i, y_i)$ is the differentiable convex loss function measuring prediction error,
- $\Omega(f_k)$ is the regularization term that penalizes model complexity.

The regularization term is defined as:

$$\Omega(f_k) = \gamma T_k + \frac{1}{2}\lambda \|w_k\|^2,$$

where T_k is the number of leaf nodes in tree k, w_k are the leaf weights, γ controls the penalty for the number of leaves, and λ is the L2 regularization parameter on leaf weights. This regularization prevents overfitting by promoting simpler tree structures.

4.5.3 Second-Order Taylor Approximation

XGBoost uses a second-order Taylor expansion of the loss function to efficiently determine optimal tree structures. At iteration t, the objective becomes:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t),$$

where:

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}},$$
$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}.$$

These first and second derivatives (gradient and Hessian) guide the tree-building process, allowing XGBoost to efficiently find optimal split points and leaf values.

4.5.4 Parameter Estimation for DGAM with XGBoost

For Double Generalized Additive Models, XGBoost is applied in an alternating fashion to model both the mean and dispersion structures. The mean and dispersion are represented as boosted ensembles:

$$\eta_{\mu,i} = F_\mu(x_i) = \sum_{k=1}^{M_\mu} f_{\mu,k}(x_i),$$
$$\eta_{\phi,i} = F_\phi(x_i) = \sum_{k=1}^{M_\phi} f_{\phi,k}(x_i),$$

where $F_\mu(\cdot)$ and $F_\phi(\cdot)$ are ensembles of M_μ and M_ϕ trees, respectively, and $f_{\mu,k}$ and $f_{\phi,k}$ are individual tree functions.

The mean and dispersion parameters are obtained via inverse link functions:

$$\mu_i = g_\mu^{-1}(\eta_{\mu,i}) = g_\mu^{-1}(F_\mu(x_i)),$$
$$\phi_i = g_\phi^{-1}(\eta_{\phi,i}) = g_\phi^{-1}(F_\phi(x_i)).$$

4.5.5 Algorithm: XGBoost-DGAM via Alternating Second-Order Boosting

The estimation of DGAM parameters using XGBoost proceeds through alternating second-order boosting for location and dispersion, as described in Algorithm 3.

Algorithm 3 XGBoost-DGAM via Alternating Second-Order Boosting for Location and Dispersion

Input: $\{(y_i, x_i)\}_{i=1}^n$, link functions g_μ, g_ϕ, distribution $p(y \mid \mu, \phi)$
Hyperparameters: Mean-stage trees M_μ, dispersion-stage trees M_ϕ, tree depth, subsampling rates, regularization parameters (γ, λ), learning rates ν_μ, ν_ϕ, stopping tolerance ε
Trainable parameters: Boosted ensembles $F_\mu(\cdot), F_\phi(\cdot)$ with $\eta_\mu = F_\mu(x)$, $\eta_\phi = F_\phi(x)$
Initialize $F_\mu^{(0)}(x) \equiv c_\mu$ and $F_\phi^{(0)}(x) \equiv c_\phi$
repeat
  Compute $\eta_{\mu,i} = F_\mu(x_i)$, $\eta_{\phi,i} = F_\phi(x_i)$
  Update $\mu_i = g_\mu^{-1}(\eta_{\mu,i})$, $\phi_i = g_\phi^{-1}(\eta_{\phi,i})$
  Mean update: Compute first and second derivatives
  $g_{\mu,i} = \frac{\partial}{\partial \eta_{\mu,i}} \left[-\log p(y_i \mid \mu_i, \phi_i) \right]$
  $h_{\mu,i} = \frac{\partial^2}{\partial \eta_{\mu,i}^2} \left[-\log p(y_i \mid \mu_i, \phi_i) \right]$
  Fit next mean tree t_μ by minimizing the quadratic approximation:
    $\mathcal{L}_\mu^{(t)} = \sum_{i=1}^n \left[g_{\mu,i} t_\mu(x_i) + \frac{1}{2} h_{\mu,i} t_\mu^2(x_i) \right] + \Omega(t_\mu)$
  Update $F_\mu \leftarrow F_\mu + \nu_\mu t_\mu$ and recompute μ_i
  Dispersion update: Compute first and second derivatives conditional on updated mean
  $g_{\phi,i} = \frac{\partial}{\partial \eta_{\phi,i}} \left[-\log p(y_i \mid \mu_i, \phi_i) \right]$
  $h_{\phi,i} = \frac{\partial^2}{\partial \eta_{\phi,i}^2} \left[-\log p(y_i \mid \mu_i, \phi_i) \right]$
  Fit next dispersion tree t_ϕ by minimizing the quadratic approximation:
    $\mathcal{L}_\phi^{(t)} = \sum_{i=1}^n \left[g_{\phi,i} t_\phi(x_i) + \frac{1}{2} h_{\phi,i} t_\phi^2(x_i) \right] + \Omega(t_\phi)$
  Update $F_\phi \leftarrow F_\phi + \nu_\phi t_\phi$
  if improvement in held-out NLL $< \varepsilon$ (or max trees reached) **then**
    break
  end if
until convergence
Output: $\hat{\mu}(x) = g_\mu^{-1}(F_\mu(x))$ and $\hat{\phi}(x) = g_\phi^{-1}(F_\phi(x))$

Detailed Algorithm Steps:

Step 1: Initialization (Line 4) Initialize the mean and dispersion ensembles as constants: $F_\mu^{(0)}(x) = c_\mu$ and $F_\phi^{(0)}(x) = c_\phi$. For Gaussian responses, c_μ is typically set to $\bar{y}$ (sample mean) and c_ϕ to $\log(s^2)$ where s^2 is the sample variance.

Step 2: Mean Boosting (Lines 8-12) For Gaussian distribution with identity link for mean:

$$g_{\mu,i} = -\frac{y_i - \mu_i}{\phi_i},$$
$$h_{\mu,i} = \frac{1}{\phi_i}.$$

Build a regression tree t_μ that minimizes the second-order approximation of the loss, using $(g_{\mu,i}, h_{\mu,i})$ to determine optimal splits and leaf values. The optimal leaf weight for a node is:

$$w^* = -\frac{\sum_{i \in \text{leaf}} g_{\mu,i}}{\sum_{i \in \text{leaf}} h_{\mu,i} + \lambda}.$$

Step 3: Dispersion Boosting (Lines 13-17) After updating the mean, compute gradients for the dispersion. For Gaussian distribution with log link for dispersion:

$$g_{\phi,i} = -\frac{1}{2}\left[\frac{(y_i - \mu_i)^2}{\phi_i} - 1\right],$$
$$h_{\phi,i} = \frac{1}{2}.$$

Build a regression tree t_ϕ for the dispersion update using these gradients.

Step 4: Regularization and Early Stopping (Lines 18-19) The regularization term $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$ penalizes tree complexity. Monitor performance on a validation set and stop when improvement falls below tolerance ε to prevent overfitting.

This alternating second-order boosting approach allows XGBoost to efficiently learn complex nonlinear relationships in both mean and dispersion while maintaining computational efficiency through the use of gradient and Hessian information.

4.6 Random Forest

Random Forest (RF), introduced by Breiman (2001), is a powerful ensemble learning method that improves prediction accuracy and robustness by combining multiple decision trees. Unlike single decision trees that can suffer from high variance and overfitting, Random Forest creates a collection of diverse trees through bootstrap sampling and random feature selection, then aggregates their predictions to produce stable and accurate results.

4.6.1 Random Forest Model Structure

A Random Forest consists of a collection of tree-based predictors, denoted as $\{f(\cdot, \Theta_k) : k = 1, 2, \ldots, K\}$, where Θ_k represents independent and identically distributed random vectors that govern the construction of each tree. For regression problems, the Random Forest prediction is obtained by averaging the predictions of all trees:

$$\hat{y}(x) = \frac{1}{K} \sum_{k=1}^{K} f(x, \Theta_k),$$

where K is the number of trees in the forest and $f(x, \Theta_k)$ is the prediction from the k-th tree.

For classification problems, the Random Forest employs a majority voting scheme, where each tree casts a vote and the class with the most votes is selected:

$$\hat{y}_{\text{class}}(x) = \underset{j \in \{1,\ldots,J\}}{\arg\max} \sum_{k=1}^{K} \mathbb{I}[f(x, \Theta_k) = j],$$

where J is the number of classes and $\mathbb{I}[\cdot]$ is the indicator function.

4.6.2 Random Forest Algorithm

The Random Forest algorithm constructs an ensemble of decision trees through the following process:

Input: A dataset $S = \{(y^1, x^1), \ldots, (y^n, x^n)\}$, where for classification $y \in \{1, \ldots, J\}$ or for regression $y \in \mathbb{R}$, and $x \in \mathbb{R}^p$ is in p-dimensional feature space. Hyperparameters include the number of trees K, the number of randomly selected features at each split d (typically $d \approx \sqrt{p}$ for classification or $d \approx p/3$ for regression), and the minimum leaf size ℓ.

Tree Construction (for $k = 1, 2, \ldots, K$):

1. Create a bootstrap sample S_k by randomly selecting n observations from the training set S with replacement. Approximately 63% of unique observations will be included, with the remaining 37% forming the out-of-bag (OOB) sample.

2. At each node of the tree:

   - Randomly select d input variables from $\{X_1, \ldots, X_p\}$
   - Based on these d variables and the data S_k at this node, determine the best split to partition the data (e.g., minimizing mean squared error for regression or Gini impurity for classification)

3. Grow the tree recursively until each terminal (leaf) node contains no more than ℓ training samples. No pruning is performed.

4. Let $f(\cdot, S_k)$ represent the constructed tree.

Output: For regression, the Random Forest predictor is:

$$f_{\text{RF}}(x, S) = \frac{1}{K} \sum_{k=1}^{K} f(x, S_k).$$

For classification, the Random Forest classifier is:

$$f_{\text{RF}}(x, S) = \underset{j \in \{1,\ldots,J\}}{\arg\max} \sum_{k=1}^{K} \mathbb{I}[f(x, S_k) = j].$$

The diversity among trees, created through bootstrap sampling and random feature selection, reduces variance while maintaining low bias, resulting in excellent predictive performance (Breiman, 2001; Hastie et al., 2009).

4.6.3 Parameter Estimation for DGAM with Random Forest

For Double Generalized Additive Models, Random Forest is applied in an alternating fashion to model both the mean and dispersion structures. The mean and dispersion are represented as forest predictors:

$$\eta_{\mu,i} = R_\mu(x_i) = \frac{1}{N_\mu} \sum_{k=1}^{N_\mu} f_{\mu,k}(x_i),$$
$$\eta_{\phi,i} = R_\phi(x_i) = \frac{1}{N_\phi} \sum_{k=1}^{N_\phi} f_{\phi,k}(x_i),$$

where $R_\mu(\cdot)$ and $R_\phi(\cdot)$ are Random Forest ensembles with N_μ and N_ϕ trees, respectively, and $f_{\mu,k}$ and $f_{\phi,k}$ are individual regression trees.

The mean and dispersion parameters are obtained via inverse link functions:

$$\mu_i = g_\mu^{-1}(\eta_{\mu,i}) = g_\mu^{-1}(R_\mu(x_i)),$$
$$\phi_i = g_\phi^{-1}(\eta_{\phi,i}) = g_\phi^{-1}(R_\phi(x_i)).$$

4.6.4 Algorithm: Random Forest DGAM via Alternating Residual Forests

The estimation of DGAM parameters using Random Forest proceeds through alternating residual forests for mean and dispersion, as described in Algorithm 4.

Algorithm 4 Random Forest DGAM via Alternating Residual Forests for Mean and Dispersion

1: **Input:** $\{(y_i, x_i)\}_{i=1}^n$, link functions g_μ, g_ϕ, distribution $p(y \mid \mu, \phi)$
2: **Hyperparameters:** Forest sizes (N_μ, N_ϕ), max depth, min leaf size, bootstrap scheme, stability constant $\delta > 0$, stopping tolerance ε
3: **Trainable parameters:** Forest predictors $R_\mu(\cdot), R_\phi(\cdot)$ with $\eta_\mu = R_\mu(x)$, $\eta_\phi = R_\phi(x)$
4: Fit initial mean forest $R_\mu^{(0)}$ on $\{(x_i, y_i)\}_{i=1}^n$ and set $\eta_{\mu,i}^{(0)} \leftarrow R_\mu^{(0)}(x_i)$
5: Initialize dispersion predictor $R_\phi^{(0)}(x) \equiv c_\phi$
6: **repeat**
7: Update $\mu_i \leftarrow g_\mu^{-1}(\eta_{\mu,i})$ and $\phi_i \leftarrow g_\phi^{-1}(R_\phi(x_i))$
8: Compute mean residuals $r_{\mu,i} \leftarrow y_i - \mu_i$ (or NLL pseudo-residuals)
9: Refit/update mean forest R_μ on $\{(x_i, r_{\mu,i})\}_{i=1}^n$
10: Update $\eta_{\mu,i} \leftarrow \eta_{\mu,i} + R_\mu(x_i)$
11: Recompute μ_i using the updated mean predictor
12: Construct dispersion targets $z_{\phi,i} \leftarrow \log\left((y_i - \mu_i)^2 + \delta\right)$
13: Fit/update dispersion forest R_ϕ on $\{(x_i, z_{\phi,i})\}_{i=1}^n$
14: **if** parameter changes (or held-out NLL change) $< \varepsilon$ **then**
15: **break**
16: **end if**
17: **until** convergence
18: **Output:** $\hat{\mu}(x) = g_\mu^{-1}(\eta_\mu(x))$ and $\hat{\phi}(x) = g_\phi^{-1}(R_\phi(x))$

Detailed Algorithm Steps:

Step 1: Initialization (Lines 4-5) Fit an initial Random Forest $R_\mu^{(0)}$ directly to the response variable y_i to obtain initial mean predictions. For Gaussian responses with identity link, this provides $\eta_{\mu,i}^{(0)} = R_\mu^{(0)}(x_i)$.

Initialize the dispersion predictor as a constant: $R_\phi^{(0)}(x) = c_\phi$, where $c_\phi = \log(s^2)$ and s^2 is the sample variance of residuals from the initial mean forest.

Step 2: Mean Forest Update (Lines 7-10) Compute residuals from the current mean predictions:

$$r_{\mu,i} = y_i - \mu_i.$$

For Gaussian DGAM, these are the simple residuals. Alternatively, negative log-likelihood pseudo-residuals can be used:

$$r_{\mu,i} = \frac{y_i - \mu_i}{\phi_i}.$$

Fit a new Random Forest to predict these residuals, then update the mean predictor by adding the residual predictions. This iterative residual fitting helps refine the mean model.

Step 3: Dispersion Forest Update (Lines 11-12) After updating the mean, construct dispersion targets based on squared residuals:

$$z_{\phi,i} = \log\left((y_i - \mu_i)^2 + \delta\right),$$

where the small constant $\delta > 0$ (e.g., $\delta = 0.01$) ensures numerical stability when residuals are very small. The logarithm is used because the dispersion model employs a log link.

Fit a Random Forest R_ϕ to predict these log-squared-residuals, which provides estimates of $\log(\phi_i)$ that can vary with covariates.

Step 4: Convergence Check (Lines 13-14) Monitor convergence by tracking changes in parameter estimates or the negative log-likelihood on a held-out validation set. Stop when changes fall below tolerance ε to prevent overfitting.

4.7 Support Vector Regression

Support Vector Regression (SVR), introduced by Vapnik (1995), is a powerful machine learning algorithm based on statistical learning theory that extends the principles of Support Vector Machines to regression problems. SVR aims to find a function that approximates the relationship between input features and continuous output values while maintaining a balance between model complexity and prediction accuracy (Smola and Schölkopf, 2004).

4.7.1 SVR Formulation

Unlike traditional regression methods that minimize the sum of squared errors, SVR employs an ε-insensitive loss function that ignores errors smaller than a threshold ε. This approach creates a margin of tolerance around the regression function, focusing computational effort on predictions that deviate significantly from observed values. The algorithm identifies support vectors—data points that lie outside or on the boundary of

this ε-tube—which are critical in determining the regression function.

The SVR optimization problem can be formulated as follows. Given a training dataset $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, SVR seeks to find a function:

$$f(x) = \langle w, \phi(x) \rangle + b,$$

where w is the weight vector, $\phi(x)$ maps input features to a higher-dimensional feature space, and b is the bias term. The optimization objective is:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*),$$

subject to the constraints:

$$y_i - \langle w, \phi(x_i) \rangle - b \leq \varepsilon + \xi_i,$$

$$\langle w, \phi(x_i) \rangle + b - y_i \leq \varepsilon + \xi_i^*,$$

$$\xi_i, \xi_i^* \geq 0,$$

where $C > 0$ is a regularization parameter that controls the trade-off between model flatness and tolerance for deviations larger than ε, and ξ_i, ξ_i^* are slack variables that allow some points to fall outside the ε-tube.

4.7.2 Kernel Methods and Dual Formulation

The dual formulation of this optimization problem involves Lagrange multipliers α_i and α_i^*, leading to the prediction function:

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(x_i, x) + b,$$

where $K(x_i, x) = \langle \phi(x_i), \phi(x) \rangle$ is a kernel function. Common kernel choices include:

- Linear kernel: $K(x_i, x_j) = \langle x_i, x_j \rangle$
- Polynomial kernel: $K(x_i, x_j) = (\gamma\langle x_i, x_j \rangle + r)^d$
- Radial Basis Function (RBF) kernel: $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$
- Sigmoid kernel: $K(x_i, x_j) = \tanh(\gamma\langle x_i, x_j \rangle + r)$

The kernel function allows SVR to capture nonlinear relationships by implicitly mapping data to higher-dimensional spaces without explicitly computing the transformation $\phi(x)$, a technique known as the "kernel trick."

4.7.3 SVR Model Structure

For a given input x, the SVR prediction is obtained as a weighted sum of kernel evaluations with the support vectors:

$$\hat{y}(x) = \sum_{i \in SV} (\alpha_i - \alpha_i^*) K(x_i, x) + b,$$

where SV denotes the set of support vectors (observations for which $\alpha_i \neq 0$ or $\alpha_i^* \neq 0$). The sparsity of the solution—only support vectors contribute to predictions—is a key computational advantage of SVR.

4.7.4 Parameter Estimation for DGAM with SVR

For Double Generalized Additive Models, SVR is applied separately to model both the mean and dispersion structures. The mean and dispersion are represented as:

$$\eta_{\mu,i} = f_\mu^{SVR}(x_i) = \sum_{j \in SV_\mu} (\alpha_{\mu,j} - \alpha_{\mu,j}^*) K_\mu(x_j, x_i) + b_\mu,$$
$$\eta_{\phi,i} = f_\phi^{SVR}(x_i) = \sum_{j \in SV_\phi} (\alpha_{\phi,j} - \alpha_{\phi,j}^*) K_\phi(x_j, x_i) + b_\phi,$$

where $f_\mu^{SVR}(\cdot)$ and $f_\phi^{SVR}(\cdot)$ are SVR models with potentially different kernel functions and hyperparameters, SV_μ and SV_ϕ are the sets of support vectors for mean and dispersion models, respectively.

The mean and dispersion parameters are obtained via inverse link functions:

$$\mu_i = g_\mu^{-1}(\eta_{\mu,i}) = g_\mu^{-1}(f_\mu^{SVR}(x_i)),$$
$$\phi_i = g_\phi^{-1}(\eta_{\phi,i}) = g_\phi^{-1}(f_\phi^{SVR}(x_i)).$$

4.7.5 Algorithm: SVR-DGAM via Alternating Optimization

The estimation of DGAM parameters using SVR proceeds through alternating optimization of mean and dispersion models, as described in Algorithm 5.

Algorithm 5 SVR-DGAM via Alternating Optimization for Mean and Dispersion

Input: $\{(y_i, x_i)\}_{i=1}^n$, link functions g_μ, g_ϕ, distribution $p(y \mid \mu, \phi)$
Hyperparameters: Regularization parameters (C_μ, C_ϕ), ε-tube widths $(\varepsilon_\mu, \varepsilon_\phi)$, kernel functions (K_μ, K_ϕ) with parameters $(\gamma_\mu, \gamma_\phi)$, convergence tolerance δ
Trainable parameters: SVR predictors $f_\mu^{SVR}(\cdot), f_\phi^{SVR}(\cdot)$ with $\eta_\mu = f_\mu^{SVR}(x)$, $\eta_\phi = f_\phi^{SVR}(x)$
Fit initial mean SVR model $f_\mu^{SVR,(0)}$ on $\{(x_i, y_i)\}_{i=1}^n$ and compute $\eta_{\mu,i}^{(0)} \leftarrow f_\mu^{SVR,(0)}(x_i)$
Initialize dispersion: compute initial residuals $e_i = y_i - g_\mu^{-1}(\eta_{\mu,i}^{(0)})$
Set initial dispersion targets $z_{\phi,i} \leftarrow \log(e_i^2 + 0.01)$
Fit initial dispersion SVR model $f_\phi^{SVR,(0)}$ on $\{(x_i, z_{\phi,i})\}_{i=1}^n$
repeat
    Compute $\eta_{\mu,i} \leftarrow f_\mu^{SVR}(x_i)$ and $\eta_{\phi,i} \leftarrow f_\phi^{SVR}(x_i)$
    Update $\mu_i \leftarrow g_\mu^{-1}(\eta_{\mu,i})$ and $\phi_i \leftarrow g_\phi^{-1}(\eta_{\phi,i})$
    Mean update: Compute weighted residuals
        $r_{\mu,i} \leftarrow \frac{y_i - \mu_i}{\sqrt{\phi_i}}$ (weighted by inverse dispersion)
    Fit SVR model f_μ^{SVR} on $\{(x_i, r_{\mu,i})\}_{i=1}^n$ with hyperparameters $(C_\mu, \varepsilon_\mu, K_\mu)$
    Solve: $\min_{w_\mu, b_\mu} \frac{1}{2}\|w_\mu\|^2 + C_\mu \sum_{i=1}^n (\xi_{\mu,i} + \xi_{\mu,i}^*)$
        subject to SVR constraints with ε_μ-insensitive loss
    Update mean predictor and recompute μ_i
    Dispersion update: Construct dispersion targets
        $z_{\phi,i} \leftarrow \log\left((y_i - \mu_i)^2 + 0.01\right)$
    Fit SVR model f_ϕ^{SVR} on $\{(x_i, z_{\phi,i})\}_{i=1}^n$ with hyperparameters $(C_\phi, \varepsilon_\phi, K_\phi)$
    Solve: $\min_{w_\phi, b_\phi} \frac{1}{2}\|w_\phi\|^2 + C_\phi \sum_{i=1}^n (\xi_{\phi,i} + \xi_{\phi,i}^*)$
        subject to SVR constraints with ε_ϕ-insensitive loss
    Evaluate negative log-likelihood $L = \sum_{i=1}^n -\log p(y_i \mid \mu_i, \phi_i)$
    if change in $L < \delta$ or max iterations reached **then**
        break
    end if
until convergence
Output: $\hat{\mu}(x) = g_\mu^{-1}(f_\mu^{SVR}(x))$ and $\hat{\phi}(x) = g_\phi^{-1}(f_\phi^{SVR}(x))$

Detailed Algorithm Steps:

Step 1: Initialization (Lines 4-7) Fit an initial SVR model for the mean directly to the response variable y_i. For Gaussian responses with identity link, this provides initial predictions $\eta_{\mu,i}^{(0)} = f_{\mu}^{SVR,(0)}(x_i)$.

Compute initial residuals $e_i = y_i - g_{\mu}^{-1}(\eta_{\mu,i}^{(0)})$ and construct log-squared-residuals as initial dispersion targets:

$$z_{\phi,i} = \log(e_i^2 + 0.01),$$

where the small constant 0.01 ensures numerical stability.

Fit an initial SVR model $f_{\phi}^{SVR,(0)}$ to predict these log-squared-residuals, providing initial dispersion estimates.

Step 2: Mean SVR Update (Lines 11-15) Compute weighted residuals that account for heteroscedasticity:

$$r_{\mu,i} = \frac{y_i - \mu_i}{\sqrt{\phi_i}}.$$

For Gaussian DGAM with identity link for mean, the weighting by $1/\sqrt{\phi_i}$ gives more importance to observations with smaller variance, improving efficiency.

Fit an SVR model by solving the primal optimization problem:

$$\min_{w_\mu, b_\mu, \xi_\mu, \xi_\mu^*} \frac{1}{2}\|w_\mu\|^2 + C_\mu \sum_{i=1}^{n}(\xi_{\mu,i} + \xi_{\mu,i}^*),$$

subject to:

$$r_{\mu,i} - (\langle w_\mu, \phi(x_i)\rangle + b_\mu) \leq \varepsilon_\mu + \xi_{\mu,i},$$

$$(\langle w_\mu, \phi(x_i)\rangle + b_\mu) - r_{\mu,i} \leq \varepsilon_\mu + \xi_{\mu,i}^*,$$

$$\xi_{\mu,i}, \xi_{\mu,i}^* \geq 0.$$

The dual formulation yields:

$$f_{\mu}^{SVR}(x) = \sum_{j \in SV_\mu} (\alpha_{\mu,j} - \alpha_{\mu,j}^*) K_\mu(x_j, x) + b_\mu,$$

where $\alpha_{\mu,j}$ and $\alpha_{\mu,j}^*$ are Lagrange multipliers obtained via quadratic programming.

Step 3: Dispersion SVR Update (Lines 16-20) After updating the mean, construct new dispersion targets based on current residuals:

$$z_{\phi,i} = \log\big((y_i - \mu_i)^2 + 0.01\big).$$

Fit an SVR model to predict these log-squared-residuals by solving:

$$\min_{w_\phi, b_\phi, \xi_\phi, \xi_\phi^*} \frac{1}{2}\|w_\phi\|^2 + C_\phi \sum_{i=1}^{n} (\xi_{\phi,i} + \xi_{\phi,i}^*),$$

subject to analogous ε_ϕ-insensitive constraints.

The resulting dispersion predictor is:

$$f_\phi^{SVR}(x) = \sum_{j \in SV_\phi} (\alpha_{\phi,j} - \alpha_{\phi,j}^*) K_\phi(x_j, x) + b_\phi.$$

For Gaussian DGAM with log link for dispersion, this directly provides $\log(\phi_i)$ estimates.

Step 4: Kernel Function Selection

The choice of kernel function $K(\cdot, \cdot)$ is crucial for SVR performance. For both mean and dispersion models, common choices include:

Radial Basis Function (RBF) kernel:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right),$$

where $\gamma > 0$ controls the kernel width. RBF kernels are effective for capturing smooth nonlinear relationships and are often the default choice.

Polynomial kernel:

$$K(x_i, x_j) = (\gamma \langle x_i, x_j \rangle + r)^d,$$

where d is the polynomial degree. This is useful when relationships are believed to be polynomial in nature.

Step 5: Hyperparameter Tuning

The performance of SVR-DGAM depends critically on hyperparameter selection:

- **Regularization parameters** (C_μ, C_ϕ): Larger values allow more training errors but may lead to overfitting; smaller values enforce stronger regularization.
- **ε-tube widths** $(\varepsilon_\mu, \varepsilon_\phi)$: Larger values create wider tubes, ignoring more small errors and producing sparser solutions.
- **Kernel parameters** $(\gamma_\mu, \gamma_\phi)$: For RBF kernels, smaller γ creates smoother functions; larger γ allows more local variation.

These hyperparameters are typically optimized using cross-validation or grid search procedures.

Step 6: Convergence Check (Lines 21-23) Monitor the negative log-likelihood:

$$L = \sum_{i=1}^{n} - \log p(y_i \mid \mu_i, \phi_i).$$

For Gaussian distribution:

$$L = \sum_{i=1}^{n} \left[\frac{1}{2} \log(2\pi\phi_i) + \frac{(y_i - \mu_i)^2}{2\phi_i} \right].$$

Terminate when the change in L falls below tolerance δ or maximum iterations are reached.

4.8 Results (Phase II): Model Implementation and Empirical Validation

Phase II operationalized the proposed Double Generalized Additive Model (DGAM) algorithms within a unified computational framework and evaluated them against the conventional Generalized Additive Model (GAM) across all experimental tables and train–validation–test splits. The primary objective of Phase II was not only to assess point prediction accuracy, but also to verify whether explicitly modeling dispersion improves probabilistic fit and uncertainty quantification in settings where heteroscedasticity is expected.

Dataset heterogeneity and variable structure. This study focuses on four publicly available healthcare datasets from Kaggle:

1. **Fetal Health Monitoring Dataset:** 2,126 cardiotocography records with 21 features for fetal health assessment during pregnancy and labor.
2. **Life Style Data:** Lifestyle and behavioral information including physical activities, dietary patterns, and physiological measurements.

3. **Health Checkup Result Dataset:** Clinical examination results from approximately 19,000,000 individuals in South Korea (2002–2020), with a stratified sample used for computational feasibility.

4. **Life Expectancy (WHO) Fixed Dataset:** Country-level health indicators for 193 countries (2000–2015) with 22 features including immunization, mortality, and socioeconomic factors.

4.8.1 Target Variables

Sixteen target variables were selected across four datasets (four per dataset):

Table 4.1 Target Variables by Dataset and Life Cycle Stage.

Stage	ID	Variable Name	Description
Birth	1.1	Abnormal_STV	Percentage of time with abnormal short-term variability
	1.2	Abnormal_LTV	Percentage of time with abnormal long-term variability
	1.3	Mean_STV	Mean value of short-term variability
	1.4	Mean_LTV	Mean value of short-term variability
Aging	2.1	Calories_Burned	Total calories burned during activities
	2.2	Fat_Percentage	Body fat percentage
	2.3	Avg_BPM	Average heart rate (beats per minute)
	2.4	Max_BPM	Maximum heart rate (beats per minute)
Illness	3.1	Creatinine	Serum creatinine (kidney function)
	3.2	GGT	Gamma-glutamyl transferase (liver enzyme)
	3.3	SGOT	Serum glutamic-oxaloacetic transaminase (AST)
	3.4	SGPT	Serum glutamic-pyruvic transaminase (ALT)
Death	4.1	Life_expectancy	Average life expectancy in years
	4.2	Adult_mortality	Adult mortality rate per 1,000 population
	4.3	Under_five_deaths	Deaths under age five per 1,000 births
	4.4	Infant_deaths	Infant deaths per 1,000 live births

4.8.2 Modeling Techniques

Five techniques were employed for modeling the mean and dispersion functions:

1. **Spline:** Smooth spline with ridge regularization
2. **EBM:** Explainable Boosting Machine with cyclic gradient boosting
3. **XGB:** Extreme Gradient Boosting with tree-based learners
4. **RF:** Random Forest ensemble method
5. **SVR:** Support Vector Regression with RBF kernel

4.8.3 Data Splitting Ratios

Three splitting ratios were investigated:

- Split A: Training (60%), Validation (20%), Test (20%)
- Split B: Training (70%), Validation (15%), Test (15%)
- Split C: Training (80%), Validation (10%), Test (10%)

Linking Phase I to Phase II. Phase I motivated the Double Generalized Additive Model (DGAM) by arguing that a mean-only additive predictor can be insufficient when dispersion varies systematically with covariates, i.e., under heteroscedasticity. Phase II operationalizes these theoretical claims through an end-to-end Python implementation and evaluates DGAM empirically across multiple healthcare datasets and experimental splits. Consequently, the Phase II results are interpreted as a direct test of the Phase I hypothesis: explicitly modeling dispersion should improve distributional fit and uncertainty quantification, even when improvements in point prediction accuracy are not guaranteed.

4.8.4 Experimental Grid and Paired-Comparison Protocol

The Phase II benchmark comprises 16 prediction tasks (four targets per dataset across four datasets), each evaluated under three train–validation–test splits (60/20/20,

70/15/15, 80/10/10) and five mean-function learners. This design yields $16 \times 3 \times 5 = 240$ matched configurations. For each configuration, GAM is fitted under a constant-dispersion assumption. DGAM is fitted by jointly learning (i) a mean submodel and (ii) a dispersion submodel. To ensure a fair one-to-one comparison, DGAM is represented by the dispersion learner with minimum negative log-likelihood (NLL) within each matched configuration. Win rates are reported as the proportion of matched configurations in which DGAM outperforms GAM on the corresponding metric.

4.8.5 Overall Comparative Performance: DGAM versus GAM

Table 4.2 reports overall win rates across all $n = 240$ paired comparisons. The most salient result is that DGAM substantially improves probabilistic fit. DGAM outperforms GAM on NLL in 90.0% of comparisons (216/240) and on CRPS in 90.8% (218/240), providing strong empirical evidence that explicit dispersion modeling yields a systematically better predictive distribution in the evaluated settings.

Calibration and coverage improve to a moderate extent. DGAM attains better empirical 95% prediction interval coverage (PICP95) in 56.7% of comparisons (136/240). This indicates that DGAM more frequently produces uncertainty intervals whose empirical coverage is closer to the nominal level than GAM, although not uniformly so across all tasks.

In contrast, point-prediction accuracy is not improved in the majority of cases. DGAM improves MAE in 6.2% of comparisons (15/240), RMSE in 7.1% (17/240), and R^2 in 7.9% (19/240). This divergence between probabilistic and point metrics is consistent with the methodological intent of DGAM: the principal benefit of explicitly learning dispersion is expected to manifest in likelihood-based criteria and proper scoring rules rather than in uniformly lower point-error metrics.

Finally, DGAM improves AIC in 75.0% of comparisons (180/240). Since AIC penalizes model complexity, this result suggests that DGAM's likelihood gains typically outweigh the additional flexibility introduced by the dispersion component.

Table 4.2 Overall win-rate of DGAM relative to GAM across all paired comparisons ($n = 240$). For AIC, NLL, CRPS, MAE, and RMSE, lower is better. For R^2 and PICP95, higher is better.

Metric	DGAM better	Win rate (%)
AIC ↓	180/240	75.0
NLL ↓	216/240	90.0
MAE ↓	15/240	6.2
RMSE ↓	17/240	7.1
R^2 ↑	19/240	7.9
CRPS ↓	218/240	90.8
PICP95 ↑	136/240	56.7

4.8.6 Robustness Across Splitting Ratios

To assess whether conclusions depend on the train–validation–test ratio, Table 4.3 reports win rates stratified by split (each split contributes $n = 80$ comparisons). The probabilistic advantage of DGAM is stable across splits. DGAM wins on NLL in 87.5–95.0% of comparisons and on CRPS in 88.8–95.0%. Coverage improvements are also consistent, with PICP95 win rates ranging from 52.5% to 58.8%.

By contrast, point-metric improvements remain low under all splits (typically below 12%), reinforcing that the empirical benefit of DGAM is primarily distributional rather than purely point-accuracy driven. Overall, the split-stratified analysis supports robustness: the key Phase II conclusion does not appear to be an artifact of a particular partitioning regime.

Table 4.3 DGAM win-rate (%) relative to GAM by train–validation–test split (each split has $n = 80$ paired comparisons).

Split	AIC ↓	NLL ↓	MAE ↓	RMSE ↓	R^2 ↑	CRPS ↓	PICP95 ↑
60/20/20	77.5	87.5	10.0	11.2	11.2	88.8	52.5
70/15/15	78.8	95.0	5.0	5.0	6.2	95.0	58.8
80/10/10	68.8	87.5	3.8	5.0	6.2	88.8	58.8

4.8.7 Heterogeneity Across Datasets

Table 4.4 reports win rates stratified by dataset (each dataset contributes $n = 60$ comparisons). DGAM improves NLL and CRPS in the majority of comparisons across all datasets (NLL: 83.3–96.7%; CRPS: 80.0–100.0%), suggesting that the benefit of explicit dispersion modeling generalizes across distinct healthcare prediction settings.

However, improvements in point-prediction metrics concentrate primarily in Dataset 1, whereas Datasets 2–4 show limited to no systematic advantage in MAE/RMSE/R^2. This result supports a nuanced interpretation: DGAM primarily enhances uncertainty modeling and distributional fit, while gains in mean accuracy are context dependent and may emerge only in subsets of tasks where improved dispersion modeling indirectly stabilizes mean estimation.

Table 4.4 DGAM win-rate (%) relative to GAM by dataset (each dataset has $n = 60$ paired comparisons).

Dataset	AIC ↓	NLL ↓	MAE ↓	RMSE ↓	R^2 ↑	CRPS ↓	PICP95 ↑
1	80.0	96.7	25.0	28.3	31.7	95.0	85.0
2	43.3	86.7	0.0	0.0	0.0	88.3	75.0
3	83.3	83.3	0.0	0.0	0.0	80.0	0.0
4	93.3	93.3	0.0	0.0	0.0	100.0	66.7

4.8.8 Most Frequently Selected Dispersion Technique Under DGAM

To interpret the empirical behavior of the dispersion component, Table 4.5 summarizes which dispersion learner is selected most frequently when DGAM is chosen by minimum NLL within each matched configuration. Random Forest (RF) and Support Vector Regression (SVR) dominate the selections (28.3% and 27.5%, respectively), followed by Explainable Boosting Machines (EBM, 21.7%). Spline-based and XGBoost-based dispersion models are selected less frequently (11.7% and 10.8%).

This selection pattern is consistent with the hypothesis that heteroscedasticity in real-world healthcare datasets may involve nonlinear and interaction-driven variance structures; flexible dispersion learners may therefore be better suited to capture such patterns and improve likelihood-based fit.

Table 4.5 Frequency with which each dispersion technique is selected as the best DGAM dispersion learner (selection by minimum NLL).

Dispersion technique	Count	Share (%)
Spline	28	11.7
EBM	52	21.7
XGB	26	10.8
RF	68	28.3
SVR	66	27.5

4.8.9 Magnitude of Improvements and Representative Examples

While win rates quantify how often DGAM outperforms GAM, they do not convey the magnitude of the improvements. To complement the win-rate analysis, Table 4.6 summarizes effect sizes of the paired differences $\Delta = \text{DGAM} - \text{GAM}$ across all $n = 240$ matched comparisons. For likelihood-based fit, DGAM exhibits sizable gains: the mean paired difference is $\Delta\text{NLL} = -232.950$ and the median is $\Delta\text{NLL} = -88.698$, indicating that improvements are not driven solely by a small number of extreme cases. Similarly, the mean and median differences are negative for CRPS (mean $\Delta\text{CRPS} = -0.479$, median $\Delta\text{CRPS} = -0.172$) and AIC (mean $\Delta\text{AIC} = -424.708$, median $\Delta\text{AIC} = -138.718$), consistent with the overall conclusion that explicit dispersion learning primarily improves distributional fit.

In contrast, point-accuracy differences concentrate near zero. The median differences for MAE and RMSE are exactly zero, and the interquartile ranges also collapse at zero for these metrics, reflecting that DGAM frequently leaves point-prediction performance essentially unchanged relative to GAM. For interval calibration, the average change in 95% coverage is close to zero (mean $\Delta\text{PICP95} \approx 0$; median $\Delta\text{PICP95} = 0.005$), aligning with the earlier win-rate finding that coverage improvements occur more often than not but are not universal.

To illustrate where the distributional improvements are most pronounced, Table 4.7 reports representative high-impact configurations selected by the largest improvements in NLL (most negative ΔNLL). These examples demonstrate that DGAM can substantially reduce NLL, often accompanied by meaningful reductions in CRPS, while leav-

ing RMSE unchanged. This empirical pattern is coherent with the DGAM design objective: improving the predictive distribution via an explicit dispersion submodel, rather than re-optimizing the conditional mean.

For completeness and methodological transparency, Table 4.8 reports representative failure-mode cases where DGAM degrades NLL relative to GAM (largest positive ΔNLL). Even in these instances, CRPS may still improve, suggesting that distributional scoring rules can disagree in particular configurations and that dispersion learning can be sensitive to model selection and data characteristics. These failure cases motivate cautious deployment practices, such as monitoring likelihood-based diagnostics and using validation-based selection of the dispersion learner, which is the strategy adopted in this study.

Overall, the effect-size summaries and representative examples provide convergent evidence for the main Phase II claim. GAM implicitly assumes constant dispersion and therefore cannot adapt when residual variability changes systematically with covariates. When heteroscedasticity is present, a mean-only model can remain competitive in point-prediction metrics, yet its predictive distribution becomes mis-specified. This mis-specification is directly penalized by NLL and AIC and is reflected by proper scoring rules such as CRPS. By explicitly learning a dispersion function, DGAM produces predictive distributions that are better fitting and more informative for uncertainty-aware decision making, which is particularly relevant in health analytics settings where downstream decisions often depend on reliable uncertainty quantification rather than point estimates alone.

Table 4.6 Effect-size summary for DGAM−GAM across all paired comparisons ($n = 240$). Negative values indicate improvements for metrics where lower is better.

Metric	n	Mean	SD	Median	Q1	Q3
AIC ↓	240	-424.708	966.985	-138.718	-810.202	-0.097
NLL ↓	240	-232.950	476.934	-88.698	-432.607	-32.191
CRPS ↓	240	-0.479	1.160	-0.172	-0.648	-0.011
PICP95 ↑ (coverage)	240	-0.000	0.065	0.005	-0.018	0.021
MAE ↓	240	-0.111	0.740	0.000	0.000	0.000
RMSE ↓	240	-0.109	0.721	0.000	0.000	0.000
R^2 ↑	240	0.010	0.056	0.000	0.000	0.000

Table 4.7 Representative high-impact examples (largest improvements in NLL; most negative ΔNLL).

Table	Dataset	Target	Split	Mean	Dispersion	NLL (GAM)	NLL (DGAM)	ΔNLL	CRPS (GAM)	CRPS (DGAM)	ΔCRPS	RMSE (GAM)	RMSE (DGAM)	ΔRMSE
9	3	var1	70/15/15	ebm	ebm	4700.368	1926.798	-2773.570	0.152	0.145	-0.007	0.805	0.805	0.000
11	3	var3	60/20/20	ebm	svr	7068.459	5722.778	-1345.681	6.636	4.486	-2.150	20.062	20.062	0.000
11	3	var3	60/20/20	xgb	svr	7034.438	5695.333	-1339.105	6.509	4.412	-2.097	19.640	19.640	0.000
15	4	var3	80/10/10	xgb	xgb	1761.777	481.344	-1280.433	1.169	1.008	-0.161	4.378	4.378	0.000
15	4	var3	80/10/10	spline	svr	1797.276	611.172	-1186.104	1.792	1.554	-0.238	5.594	5.594	0.000

Table 4.8 Representative failure-mode examples (worst ΔNLL; DGAM degrades NLL relative to GAM).

Table	Dataset	Target	Split	Mean	Dispersion	NLL (GAM)	NLL (DGAM)	ΔNLL	CRPS (GAM)	CRPS (DGAM)	ΔCRPS	RMSE (GAM)	RMSE (DGAM)	ΔRMSE
9	3	var1	60/20/20	ebm	ebm	1417.619	3975.411	2557.792	0.135	0.128	-0.007	0.483	0.483	0.000
9	3	var1	70/15/15	svr	xgb	949.700	4647.266	3697.566	0.216	0.123	-0.093	0.362	0.362	0.000