Generalized Additive Models (GAMs)

Generalized Additive Models (GAMs) are a class of statistical models used in predictive analysis that extend Generalized Linear Models (GLMs) by allowing for more flexibility in how the predictors (independent variables) relate to the response (dependent variable).
->Predictors:**(Independent Variables, Features, or Explanatory Variables)**
These are the **input** variables that you use to make predictions or to explain changes in another variable. They are also known as **independent variables** or **features** in machine learning.

Examples of predictors:

- In a house price prediction model, predictors could be:
  - Square footage of the house
  - Number of bedrooms
  - Age of the house
  - Location (neighborhood)
- In a healthcare model, predictors might include:
  - Age of the patient
  - Weight, height, and BMI
  - Smoking habits
  - Medical history (e.g., presence of diabetes)

- > Response:**(Dependent Variable or Target Variable)**:

The **response** is the **outcome** or **dependent variable** that you are trying to predict or explain using the predictors.

It represents the result or the thing you're interested in modeling, estimating, or understanding.
Examples of responses:

- In the house price prediction example, the response would be:
  - The price of the house (what you are trying to predict based on the predictors).
- In a healthcare model, the response might be:
  - Whether a patient has a particular disease (binary outcome: yes/no).
  - The cost of medical treatment.
  - The number of hospital visits a patient will make over a year.

In traditional GLMs, the relationship between predictors and the response is typically modeled as a linear function. However, GAMs allow the relationship between predictors and the response to be more flexible by using smooth functions instead of linear terms. This flexibility enables GAMs to capture non-linear patterns in data

In GLM, the relation b/w predictors(x1,x2,…,xp) and response(y) is given by:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon.$$

In order to allow for non-linear effects a GAM replaces each linear component βjxj with a smooth non-linear function fj(xj):

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_p(x_p) + \epsilon.$$

where (f1,f2,…,fp) are smooth functions, and ϵ is error rate

This is called an additive model because we estimate each fj(xj) for j=1,2,3,....,p and then add together all of these individual contributions.

Fitting Methods

The **fitting methods** in Generalized Additive Models (GAMs) are designed to estimate the smooth functions f1(x1),f2(x2),…,fp(xp)f_1(x_1), f_2(x_2), \ldots, f_p(x_p)f1(x1),f2(x2),…,fp(xp) that represent the non-linear effects of the predictors.

Backfitting Algorithm

The **backfitting algorithm** is the most commonly used approach to fit GAMs. It works iteratively to estimate each smooth function while holding the others constant.

Steps of Backfitting:

1. **Initialization**: Start with some initial guesses for each smooth function fj(xj), often setting them to zero or a constant.
2. **Iterative Updates**:
    - For each predictor xj, calculate the residuals(errors)(the differences between the actual values and the predicted values) after removing the contributions of all other predictors:

$$r_j = y - \beta_0 - \sum_{k \neq j} f_k(x_k)$$

    - Fit a smooth function fj(xj) to these residuals using a smoothing method (like splines or loess).
3. **Repeat**: Iterate over all predictors until the changes in fj(xj) are negligible.

Advantages:

- Simple and efficient.
- Works well for additive models because it treats each predictor independently.

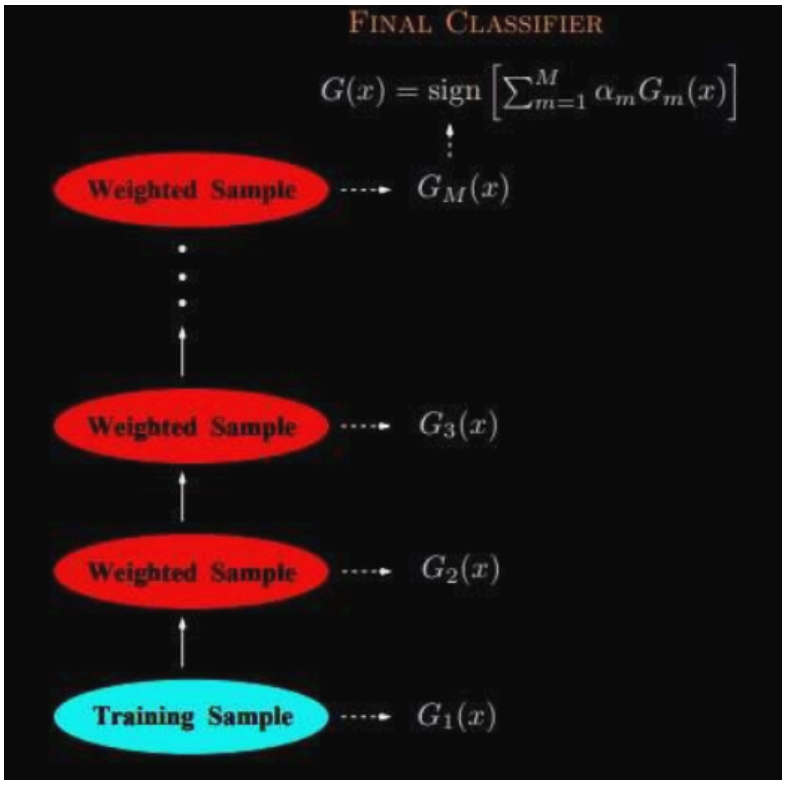
Boosting Methods

Boosting is a powerful **ensemble learning technique** used in machine learning, where multiple weak learners (typically simple models like decision stumps or shallow trees) are combined iteratively to create a strong predictive model. It works by sequentially training models, each focusing on the mistakes made by its predecessors, and combining their predictions to improve accuracy.

How Boosting Works

Boosting aims to minimize the error by iteratively refining the model. Here's a simplified **step-by-step process**:

1. Start with a weak learner (e.g., a shallow tree) trained on the data.
2. Evaluate its predictions and compute the error.
3. Update weights for the data points:
    - Increase weights for incorrectly predicted points.
    - Decrease weights for correctly predicted points.
4. Train the next weak learner on the updated weights.
5. Combine all learners' predictions (using weights that depend on their performance).



AdaBoost (Adaptive Boosting)

AdaBoost (Adaptive Boosting) is one of the first and most popular boosting algorithms.
It combines multiple **weak learners** (often decision stumps) to create a **strong learner**.
AdaBoost assigns **weights** to data points, increasing the weight of misclassified points to make them more important in subsequent iterations.
Works for **classification** problems but can also be extended to regression.

Algorithm

1. Initialization:

- Assign equal weights (wi) to all (N) training samples. $w_i = \frac{1}{N}$

2. Iterative Steps:

For m=1,2,…,M (where M is the number of weak learners):

1. Train a weak learner hm(x) on the weighted dataset.
2. Calculate the weighted error εm:

$$\varepsilon_m = \frac{\sum_{i=1}^{N} w_i \cdot 1(y_i \neq h_m(x_i))}{\sum_{i=1}^{N} w_i}$$

where $1(y_i \neq h_m(x_i))$ is 1 if $y_i \neq h_m(x_i)$, otherwise 0.

3. Compute the model weight αm:

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$

- Learners with lower error are given higher weights.

4. Update the weights of the data points:

$$w_i \leftarrow w_i \cdot e^{\alpha_m \cdot 1(y_i \neq h_m(x_i))}$$

- Normalize the weights so they sum to 1.

3. Final Model:

The final prediction is a weighted majority vote of all weak learners:

$$F(x) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m h_m(x) \right)$$

- One of the disadvantages is that Sequential training makes it slower compared to parallelizable methods like Random Forest.

Numerical Optimization via Gradient Boosting

Gradient Boosting is a powerful machine learning technique that uses **numerical optimization** to build a strong learner from an ensemble of weak learners. The key idea is to iteratively minimize a loss function by fitting new models to the residual errors of the existing ensemble.

Key Components

1. **Loss Function** $(L(y, F(x)))$:
    - Represents the difference between the true target value (y) and the model prediction (F(x)).
    - Common loss functions:
        - Mean Squared Error (MSE) for regression.
        - Log-loss (Binary Cross-Entropy) for classification.
    - Gradient Boosting optimizes this loss function directly.
2. **Gradient Descent**:
    - The method of optimization where the model iteratively minimizes the loss by moving in the direction of the negative gradient. It is an algorithm that finds the minimum value of a function
3. **Weak Learners**:
    - Typically **decision trees** with small depth (e.g., stumps).
    - At each iteration, the new weak learner is trained to approximate the gradient of the loss function.

Algorithm for Gradient Boosting

1. Initialization:

- Start with a constant model that minimizes the loss:

$$F_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

For MSE, this is simply the mean of the target values.

2. Iterative Optimization:

For $m = 1, 2, \ldots, M$:

1. **Compute Residuals (Gradients):**
    - The negative gradient of the loss function with respect to the predictions:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$$

These residuals represent the "errors" the current model needs to correct.

2. **Fit a Weak Learner:**
    - Train a weak learner $h_m(x)$ to predict the residuals:

$$h_m(x) \approx r_{im}$$

3. **Update the Model:**
    - Add the new weak learner to the ensemble with a learning rate ν:

$$F_m(x) = F_{m-1}(x) + \nu h_m(x)$$

3. Final Prediction:

After M iterations, the final model is:

$$F(x) = F_0(x) + \sum_{m=1}^{M} \nu h_m(x)$$