Model Overview

1 Classification Model Overview

1.1 Random Forest (RF)

Random Forest (RF) is an ensemble learning algorithm used primarily for classification and regression tasks. It creates a multitude of decision trees during the training phase, combining their individual predictions to improve accuracy and reduce overfitting.

1.1.1 Mathematical Formulation

Given a training dataset, Random Forest builds K decision trees, each trained independently on a randomly sampled subset of the data (with replacement), using random subsets of features at each split. The final prediction is made by aggregating the predictions of all individual trees:

For classification, the final predicted class $\hat{y}$ is the majority vote across all trees:

$$\hat{y} = \text{majority\_vote}\left(T_1(x), T_2(x), \ldots, T_K(x)\right)$$

For regression, the final predicted value is the average of the predictions:

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

Where:

- x: Input feature vector.
- K: Total number of decision trees in the forest.
- $T_k(x)$: Prediction from the k-th decision tree.
- $\hat{y}$: Final aggregated prediction.

1.1.2 Key Components of the Algorithm

- **Bootstrap Aggregating (Bagging)**: Each decision tree is trained on a random subset (sampled with replacement) from the original training data, ensuring diversity among trees.
- **Random Feature Selection**: For each split in a tree, a random subset of features is chosen. This reduces correlation between trees and leads to better generalization.
- **Aggregation of Predictions**: The individual tree predictions are combined by majority voting (classification) or averaging (regression).

1.1.3 Advantages and Characteristics

- Robustness against overfitting due to ensemble averaging.
- High predictive accuracy by aggregating multiple trees.
- Easy interpretation of feature importance.
- Capability to handle large datasets and numerous features effectively.
- Effective handling of missing values and noisy data.

1.2 XGBoost (Extreme Gradient Boosting)

XGBoost is an optimized gradient boosting algorithm designed for supervised learning tasks, including classification and regression. It builds an ensemble of decision trees sequentially, where each new tree focuses on correcting errors made by previous trees, significantly enhancing predictive performance.

1.2.1 Mathematical Formulation

XGBoost minimizes the following objective function:

$$\text{Obj}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

This objective consists of two components:

1. **Loss Function (Fit Term)**:

   $$\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))$$

   Represents the discrepancy between the true labels y_i and the predicted values after adding the current tree $f_t(x_i)$. Here, $\hat{y}_i^{(t-1)}$ is the prediction obtained from all previous trees combined, and $f_t(x_i)$ is the current tree prediction.

2. **Regularization Term**:

   $$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

   Controls model complexity to prevent overfitting:

   - γ: Penalizes the number of leaf nodes T in each tree, controlling tree complexity.
   - λ: Penalizes large leaf weights (w_j), encouraging simpler models.

The final prediction after K boosting rounds (trees) is:

$$\hat{y}_i = \sum_{t=1}^{K} f_t(x_i)$$

1.2.2 Notation and Definitions

- x_i: Feature vector for the i-th sample.
- y_i: True label for the i-th sample.
- $\hat{y}_i^{(t-1)}$: Prediction from previous $t-1$ trees for the i-th sample.
- $f_t(x_i)$: Prediction of the current tree t for the i-th sample.
- $l(y_i, \hat{y}_i)$: Loss function (e.g., squared error for regression, log-loss for classification).
- K: Total number of boosting rounds (trees).
- T: Number of leaf nodes in the current tree.
- w_j: Weight (prediction score) of leaf node j.
- γ: Regularization parameter controlling tree complexity.
- λ: Regularization parameter controlling leaf weight magnitude.

1.2.3 Advantages and Characteristics

- High predictive accuracy through sequential error correction.
- Efficient parallelization and distributed computing capabilities.
- Built-in regularization prevents overfitting.
- Automatic handling of missing values.
- Scalable and computationally efficient.

1.3 GLMNET (Elastic Net)

GLMNET is a supervised learning algorithm that fits a generalized linear model (GLM) using **Elastic Net regularization**. Elastic Net combines L1 (Lasso) and L2 (Ridge) penalties, shrinking coefficients toward zero to reduce complexity and improve model generalization.

1.3.1 Mathematical Formulation

GLMNET solves the following optimization problem:

$$\min_{\beta_0,\beta} \left[\frac{1}{N} \sum_{i=1}^{N} w_i \, \ell\left(y_i, \beta_0 + \beta^\top x_i\right) + \lambda \left(\frac{1-\alpha}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \right) \right]$$

The objective function consists of two parts:

1. **Loss Term**:
$$\frac{1}{N} \sum_{i=1}^{N} w_i \, \ell(y_i, \beta_0 + \beta^\top x_i)$$
Measures the discrepancy between the predicted values $\hat{y}_i$ and true labels y_i.

2. **Elastic Net Penalty** (Regularization Term):

$$\lambda \left(\frac{1-\alpha}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \right)$$

Combines two regularizations:

- L2 penalty ($\|\beta\|_2^2$): Shrinks coefficients toward zero, preventing large values.
- L1 penalty ($\|\beta\|_1$): Encourages sparsity, pushing some coefficients exactly to zero, thus enabling feature selection.

1.3.2 Notation and Definitions

- β_0: Intercept term (bias).
- β: Vector of coefficients corresponding to the input features.
- N: Number of training samples.
- x_i: Feature vector for the i-th sample.
- y_i: True label for the i-th sample.
- w_i: Sample weight (optional; default typically 1).
- $\ell(y_i, \hat{y}_i)$: Loss function (typically negative log-likelihood), measuring error between true and predicted values.
- λ: Regularization parameter controlling penalty strength.
- α: Elastic net mixing parameter:
  - $\alpha = 1$: Pure Lasso (L1)
  - $\alpha = 0$: Pure Ridge (L2)
  - $0 < \alpha < 1$: Elastic Net (combination)
- $\|\beta\|_1$: L1 norm (sum of absolute values of coefficients).
- $\|\beta\|_2^2$: Squared L2 norm (sum of squared coefficients).

1.3.3 Advantages and Characteristics

- Combines strengths of L1 and L2 regularization.
- Good interpretability.
- Encourages sparsity (feature selection).
- Flexible tuning through parameters λ and α.

1.3.4 Remark

This model was selected as the final model for further analysis due to its strong performance and its ease of interpretability. Its interpretability stems from maintaining a linear relationship between predictors and the response variable, which allows for straightforward interpretation of feature effects. Additionally, the Elastic Net regularization induces sparsity by shrinking less relevant coefficients toward zero, resulting in a simpler and more focused model that highlights the most important variables.

2 Clustering Model Overview

2.1 K-means Clustering

K-means clustering is an unsupervised learning algorithm that partitions a dataset into K distinct, non-overlapping clusters based on feature similarity. It aims to minimize the variance within each cluster, thus grouping similar data points together.

2.1.1 Mathematical Formulation

Given a dataset $\{x_1, x_2, \ldots, x_n\}$ where each $x_i \in \mathbb{R}^d$, K-means seeks to minimize the following objective function:

$$\underset{C}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Where:

- $C = \{C_1, C_2, \ldots, C_K\}$ is the set of clusters.
- μ_k is the centroid (mean) of cluster C_k.
- $\|x_i - \mu_k\|^2$ is the squared Euclidean distance between data point x_i and the centroid μ_k.

2.1.2 Algorithm Steps

1. Initialize K centroids randomly.
2. Assign each data point to the nearest centroid based on Euclidean distance.
3. Recompute the centroids as the mean of all data points assigned to each cluster.
4. Repeat steps 2 and 3 until convergence (no further changes in cluster assignments or centroids).

2.1.3 Advantages and Characteristics

- Simple and computationally efficient, making it scalable to large datasets.
- Works well when clusters are spherical and similarly sized.
- Sensitive to initial centroid placement; different initializations can lead to different results.
- Requires specification of the number of clusters K in advance.

2.2 Hierarchical Clustering

Hierarchical clustering is an unsupervised learning algorithm that builds a hierarchy of clusters either in a bottom-up (agglomerative) or top-down (divisive) manner. It does not require the number of clusters to be specified in advance and produces a dendrogram that visualizes the nested clustering structure.

2.2.1 Mathematical Formulation

Given a dataset $\{x_1, x_2, \ldots, x_n\}$, hierarchical clustering iteratively merges or splits clusters based on a linkage criterion, minimizing or maximizing the dissimilarity between clusters.

The linkage between two clusters A and B is defined differently depending on the chosen strategy:

- **Single linkage** (minimum distance):

$$d(A, B) = \min_{x \in A, y \in B} \|x - y\|$$

- **Complete linkage** (maximum distance):

$$d(A, B) = \max_{x \in A, y \in B} \|x - y\|$$

- **Average linkage** (average distance):

$$d(A, B) = \frac{1}{|A||B|} \sum_{x \in A} \sum_{y \in B} \|x - y\|$$

Where $\|x - y\|$ denotes the Euclidean distance between points x and y.

2.2.2 Algorithm Steps

- **Agglomerative Approach** (bottom-up):
  1. Start with each data point as a separate cluster.
  2. At each step, merge the two closest clusters based on the linkage criterion.
  3. Repeat until all points are merged into a single cluster or until a stopping condition is met.
- **Divisive Approach** (top-down):
  1. Start with all data points in a single cluster.
  2. Recursively split clusters until each cluster contains a single point or meets a stopping condition.

Common stopping conditions include reaching a predefined number of clusters, exceeding a distance threshold between clusters, or achieving a desired cluster quality based on a linkage criterion.

2.2.3 Advantages and Characteristics

- Does not require specification of the number of clusters in advance.
- Produces a dendrogram, providing rich information about the data's structure.
- Flexible with different linkage methods and distance metrics.
- Sensitive to noise and outliers, especially in the agglomerative approach.
- Computationally intensive for large datasets.

2.3 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a density-based unsupervised learning algorithm that identifies clusters as areas of high point density, separating noise points in sparse regions. It is particularly effective at discovering clusters of arbitrary shape and handling outliers.

2.3.1 Parameters

DBSCAN relies on two key parameters:

- ε (epsilon): The maximum distance between two points for one to be considered as in the neighborhood of the other.
- MinPts: The minimum number of points required to form a dense region (including the point itself).

2.3.2 Key Definitions

- **Core point**: A point with at least MinPts points (including itself) within its ε-neighborhood.
- **Directly density-reachable**: A point y is directly density-reachable from a core point x if y lies within the ε-neighborhood of x.
- **Density-reachable**: A point y is density-reachable from x if there exists a chain of points $x_1, x_2, \ldots, x_n$ with $x_1 = x$ and $x_n = y$, where each x_{i+1} is directly density-reachable from x_i.
- **Noise**: Points that are not density-reachable from any core point.

2.3.3 Algorithm Steps

1. For each unvisited point, retrieve its ε-neighborhood.
2. If the point is a core point, create a new cluster and expand it by recursively including all points that are density-reachable.
3. If the point is not a core point and not density-reachable from any other core point, label it as noise.
4. Repeat until all points have been processed.

2.3.4 Advantages and Limitations

- Can discover clusters of arbitrary shape and size.
- Robust to noise and outliers.
- Does not require prior specification of the number of clusters.
- Sensitive to the choice of ε and MinPts.
- Struggles with datasets containing clusters of varying density.

2.4 Gaussian Mixture Model (GMM)

Gaussian Mixture Model (GMM) is a probabilistic unsupervised learning algorithm that models a dataset as a mixture of several Gaussian distributions with unknown parameters. It provides a soft clustering approach where each data point is assigned a probability of belonging to each cluster.

2.4.1 Mathematical Formulation

Given a dataset $\{x_1, x_2, \ldots, x_n\}$, GMM assumes that each data point is generated from one of K Gaussian components, each defined by a mean vector μ_k, a covariance matrix Σ_k, and a mixing coefficient π_k.

The probability density function for a point x is:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

Where:

- $\mathcal{N}(x \mid \mu_k, \Sigma_k)$ denotes the multivariate Gaussian distribution:

$$\mathcal{N}(x \mid \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right)$$

  where d is the dimensionality of the data, i.e., the number of features in each data point.

- π_k is the mixing coefficient for the k-th Gaussian component, with $\sum_{k=1}^{K} \pi_k = 1$ and $0 \leq \pi_k \leq 1$.

Model parameters (π_k, μ_k, Σ_k) are typically estimated using the Expectation-Maximization (EM) algorithm.

2.4.2 Algorithm Steps

1. Initialize parameters (π_k, μ_k, Σ_k) randomly or using K-means clustering.
2. **Expectation step (E-step)**: Compute the responsibility $\gamma(z_{ik})$, the probability that data point x_i belongs to cluster k.

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

3. **Maximization step (M-step)**: Update the parameters (π_k, μ_k, Σ_k) using the computed responsibilities:
   - Update mixing coefficients:

$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} \gamma(z_{ik})$$

   - Update means:

$$\mu_k = \frac{\sum_{i=1}^{n} \gamma(z_{ik}) x_i}{\sum_{i=1}^{n} \gamma(z_{ik})}$$

   - Update covariance matrices:

$$\Sigma_k = \frac{\sum_{i=1}^{n} \gamma(z_{ik})(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} \gamma(z_{ik})}$$

4. Repeat E-step and M-step until convergence (i.e., parameters stabilize).

2.4.3 Advantages and Characteristics

- Provides soft cluster assignments (probabilistic membership).
- Can model complex cluster shapes through covariance matrices.
- More flexible than K-means, which assumes spherical clusters.
- Sensitive to initialization and may converge to local optima.
- Assumes underlying data distribution is Gaussian.

2.4.4 Remark

Gaussian Mixture Models (GMM) were excluded from the research due to their soft clustering nature, which produces probabilistic rather than fixed cluster assignments. Although GMM is capable of modeling complex, non-spherical structures, it was deemed less practical for the study's objective of creating clear and interpretable groupings for policy applications. Hard clustering methods such as K-means, hierarchical clustering, and DBSCAN were preferred for their greater interpretability in real-world contexts.

3 Forecasting Model Overview

3.1 ARIMA: AutoRegressive Integrated Moving Average

The ARIMA model is a widely used approach in time series forecasting. It is particularly effective for data that exhibit autocorrelation and non-stationarity. The acronym stands for AutoRegressive Integrated Moving Average, and the model is defined by three components:

- **AR (AutoRegressive)**: The relationship between an observation and a number of lagged observations.
- **I (Integrated)**: The differencing of raw observations to make the time series stationary.
- **MA (Moving Average)**: The dependency between an observation and a residual error from a moving average model applied to lagged observations.

Mathematically, the ARIMA(p, d, q) model is expressed as:

$$Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

Note: this equation is applied after the original time series has been differenced d times to ensure stationarity.

- Y_t is the observed value at time t (after differencing).
- c is a constant term.
- ϕ_i are the autoregressive coefficients.
- θ_j are the moving average coefficients.
- ε_t is white noise.
- d is the number of times the original series has been differenced to remove trends and achieve stationarity.

In this study, the `auto.arima()` function from the `forecast` package in R was used to automatically select the optimal model order by minimizing the Akaike Information Criterion (AIC).

3.2 Prophet: Decomposable Time Series Model

Prophet is a decomposable time series forecasting model developed by Facebook, designed to handle time series with strong trends and seasonality, even when data is missing or contains outliers. The model represents the time series as the sum of several components:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t$$

Where:

- $g(t)$ represents the trend component (linear or logistic growth),
- $s(t)$ captures periodic seasonal effects using Fourier series,
- $h(t)$ represents the effects of holidays (if specified),
- ε_t is the error term.

For this study, the dataset consists of annual observations. As a result, the seasonality term $s(t)$ was disabled. The holiday component $h(t)$ was excluded as well, since holiday effects require higher-frequency data and cannot be meaningfully captured at the annual level. Despite the absence of these components, however, Prophet remains practically useful as a flexible, trend-focused forecasting model, well suited for capturing long-term patterns. Therefore, the use of the Prophet model remains appropriate for this study. The model was implemented using the `prophet` package in R, with forecasts generated through automatic trend fitting and uncertainty estimation. Prophet is widely used in applied forecasting for its flexibility, interpretability, and robust performance on real-world data.