Comparison of Bayesian and Non-Parametric Models on Imbalanced Clinical Data

Feng Gu (T00751197), Ogunkeye Taiwo (T00751495),
Yazhe Wu (T00752976), Jiahui Yan (T00752485),
Tianle Zhong (T00749667)

Abstract

This study develops a flexible and robust framework for predicting functional disability levels using the SUPPORT2 clinical dataset, where class imbalance and nonlinearity present significant modeling challenges. The target variable (`sfdm2`) is highly skewed, with underrepresented classes severely impacting model performance. To address this, we construct a modeling pipeline that integrates spline-transformed Bayesian logistic regression with bootstrap-based prior estimation, allowing for non-linear decision boundaries and uncertainty quantification.

We compare this approach to classical Maximum Likelihood Estimation (MLE), Random Forests, and K-Nearest Neighbors (KNN), incorporating various resampling strategies including SMOTE and an augmented SMOTE with PCA. Feature selection is guided by LASSO regression, and model performance is assessed using precision, recall, F1-score, and AUC-ROC curves in both training and test sets.

Results show that the Bayesian spline-based models outperform MLE in terms of stability and minority class recognition, while ensemble methods like Random Forests achieve the highest overall accuracy. Advanced resampling with PCA improves synthetic sample quality, enabling better generalization in underrepresented classes.

The framework provides a modular, interpretable, and effective solution for multi-class classification in imbalanced healthcare datasets, with potential for clinical application in decision support systems.

The complete datasets, source code, and results presented in this report are available in the GitHub repository: `https://github.com/Gufeng-2002/5420_Final_Project`.

1 Introduction

In critical care settings, accurately predicting patient outcomes is essential for clinical decision-making. However, existing prognostic models often struggle with class imbalance issues, especially failing to identify high-risk minority subgroups such as comatose patients.

The SUPPORT2 dataset, available in the UCI Machine Learning Repository, contains clinical records of 9,105 hospitalized patients. The dataset includes a wide range of clinical features and patient disability outcomes, with the target variable (sfdm2) classifying patients into five disability levels. Nonetheless, the distribution of outcome categories in this dataset is highly imbalanced, posing significant challenges for conventional classification algorithms in detecting rare but clinically important cases.

The SUPPORT2 dataset is accessible on UCI-SUPPORT2 repository. It contains clinical data from the Study to Understand Prognoses Preferences Outcomes and Risks of Treatment (SUPPORT), a multi-center study of hospitalized adults with life-threatening conditions.

1.1 Basic Information about SUPPORT 2

The SUPPORT2 dataset contains a comprehensive set of variables capturing patient demographics, clinical status, outcomes, and treatment-related factors. a more detailed information of variables is in table 16(appendix) below is a high-level summary:

- **Identifiers:** `id` is used to uniquely identify each patient.
- **Demographics:** Includes variables such as `age`, `sex`, `race`, and `edu` (education level).
- **Outcomes:** Key target variables include `death` (death by end of study), `hospdead` (in-hospital death), and `sfdm2` (functional disability score).
- **Clinical Measurements (Day 3):** Vitals and lab results such as `meanbp` (mean arterial blood pressure), `wblc` (white blood cell count), `hrt` (heart rate), `resp` (respiration rate), `temp` (temperature), `pafi` (PaO_2/FiO_2 ratio), and blood markers including `alb`, `bili`, `crea`, `bun`, `sod`, `ph`, `glucose`, and `urine`.

- **Disease and Comorbidity:** Includes categorical disease classifications `dzgroup`, `dzclass`, `ca` (cancer status), and binary flags for `diabetes` and `dementia`. The variable `num.co` captures the number of comorbidities.
- **Functional Status:** Captured through indices such as `adlsc`, `adlp`, and `adls`, representing the ability to perform activities of daily living.
- **Severity Scores and Prognosis:** Includes clinical scoring systems like `scoma` (coma score), `aps`, `sps`, `avtisst`, and predicted survival estimates (`surv2m`, `surv6m`, `prg2m`, `prg6m`).
- **Healthcare Utilization and Cost:** Includes `charges`, `totcst` (total cost), and `totmcst` (total micro cost), reflecting resource use during hospitalization.
- **Hospitalization Timing:** `slos` (length of stay), `d.time` (follow-up time), `dnrday` (DNR order timing), and `hday` (hospital identifier).

2 Literature Review

Several papers have sought to improve classification outcomes on imbalanced datasets.

Pan et al.(2020) proposed adaptive solutions for imbalance learning using Adaptive and Gaussian Oversampling in extremely imbalanced datasets. They developed Adaptive-SMOTE, which improves regular SMOTE by adaptively selecting the minority class while preserving distribution characteristics. Although the study was robust due to the number of datasets used, its limitations include high computational complexity and reduced generalizability, especially for streaming data with dynamic imbalance ratios.

Swana et al.(2022) examined class imbalance challenges in fault classification for wound-rotor induction generators (WRIGs). They used SMOTE combined with Tomek links (T-Link) and evaluated classifiers including Naive Bayes, SVM, and KNN. Their comprehensive methodology validated performance using both simulated and experimental data. While their technique demonstrated strong applicability in industrial settings, the study was limited to WRIGs and did not address the computational cost of SMOTE.

Mohammed (2020) tackled class imbalance in diabetes diagnosis. He combined SMOTE with normalization techniques (min-max, z-score, and L2 norm), using performance metrics like accuracy, F1-score, and precision. The study contributed the ZADA dataset and used six classifiers for comparative analysis. Despite its practical significance, the dataset size was small (909 samples), collected only in Kurdistan, Iraq, which limits broader applicability. Additionally, there was no analysis of feature importance or discussion of clinical deployment.

3 Data Description and EDA

We summarized the dataset's key statistics to identify trends, central tendencies, and potential anomalies in the data. Descriptive summaries provided insights into the distribution of demographic, clinical, and hospitalization-related variables. From the Shapiro-Wilk Test shown in Table 17, we can confirmed that nearly all variables deviated significantly from normality. As we can see from Figure 10, 11 in Appendix, many of the numerical variables had heavy-tailed distributions, exhibited significant skewness and the presence of outliers. This finding suggested the need for scaling and normalization techniques to ensure reliable modeling.

To explore inter-variable relationships and assess the risk of multicollinearity, we performed a correlation analysis across the dataset. The results(Figure 1) identified strong linear relationships among several variable groups:

1) **Medical Cost Variables:** `charges`, `totcst`, and `totmcst` showed high positive correlations.

2) **Physiological Measures:** `aps`, `sps`, and `avtisst` demonstrated strong internal correlations.

3) **Survival Probabilities:** `surv2m`, `surv6m`, `prg2m`, and `prg6m` were highly correlated, suggesting they reflect overlapping prognostic information.

4) **Activities of Daily Living (ADL):** `adlp`, `adls`, and `adlsc` were also strongly related.

Understanding the distribution of our target variable `sfdm2` is crucial for selecting appropriate modeling approaches and evaluation metrics. `sfdm2` quantifies patient functional disability on a 1–5 severity scale, with 5 representing maximum impairment. This metric was derived from Sickness Impact Profile (SIP) questionnaires via patients and/or their surrogates to systematically evaluate functional status.

The proportion of 5 different target values in response variable is shown in Figure12 in Appendix. This imbalance necessitates careful consideration during model development, potentially requiring techniques such as resampling,class weighting, or specialized algorithms.

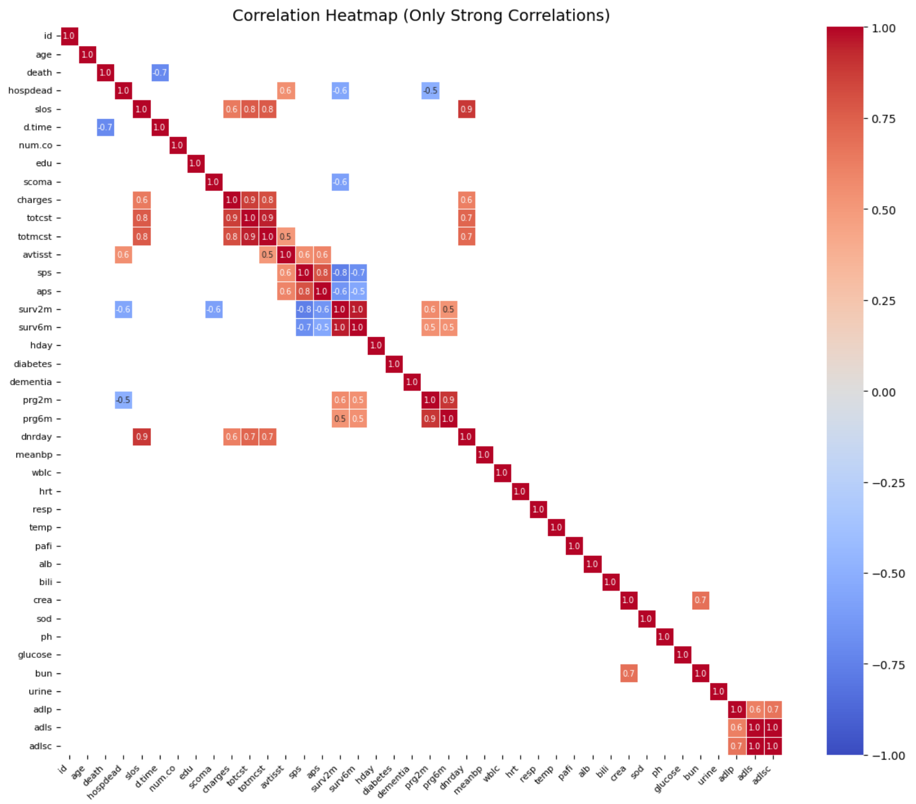


Figure 1: Correlation matrix of the SUPPORT2 dataset.

4 Data Preprocessing and feature selection

4.1 Missing Value Imputation

The dataset contains 9105 rows and 48 columns with varying degrees of missingness(Figure 2). For target variable `sfdm2`, there are 1400 missing rows, making up 15.38% of the total. These rows were removed to avoid data leakage, as imputing the target variable would undermine model integrity. While 32/47 predictor variables had missing values, with some exceeding 50% (e.g., `adlp`, `urine`).

To address missing values, we employed a method of combination of clinical imputation and custom imputation pipelines. The imputation process was designed to ensure that the data remained representative and suitable for modeling.

1. **Clinical Imputation:** We used reference values from the dataset documentation to impute 7 clinical variables:

Table 1: SFDMS Category Descriptions

Level	Class Name	Meanings
1	no (Month 2 & SIP present)	No signs of moderate to severe functional disability.
2	adl ≥ 4 (≥ 5 if survived)	Patient was unable to do 4 or more activities of daily living.
3	SIP ≥ 30	Sickness Impact Profile total score at 2 months is greater or equal to 30.
4	Coma or Intubated	Patient intubated or in coma.
5	$<$ 2 mo. follow-up	Patient died before 2 months after study entry.

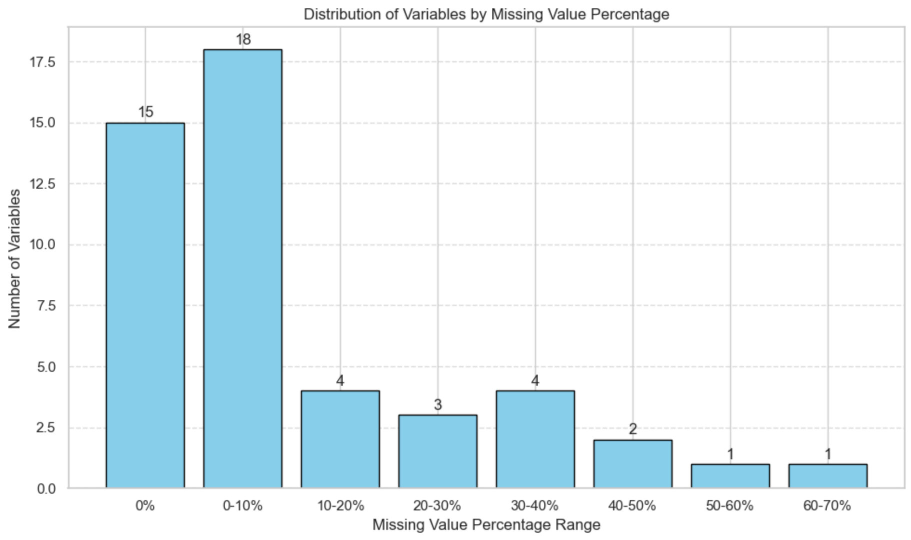


Figure 2: Distribution of missing values in the dataset

- `alb`: 3.5, `pafi`: 333.3, `bili`: 1.01, `crea`: 1.01, `bun`: 6.51, `wblc`: 9.0, `urine`: 2502

2. **Custom Imputation Pipelines:** We designed three imputer types tailored to the data types:

Table 2: Imputation Strategy Table

Imputer Type	Variable Feature	Imputation Methods
I	Numerical	Mean, Median, Constant, KNN
II	Categorical	One-hot: Most frequent, Constant `Unknown`
III	Categorical	Simple: Most frequent, Constant `Unknown`

By combining the methods in each imputer type, we generated 16 distinct imputed datasets ($4 \times 2 \times 2$) to enable comparative modeling.

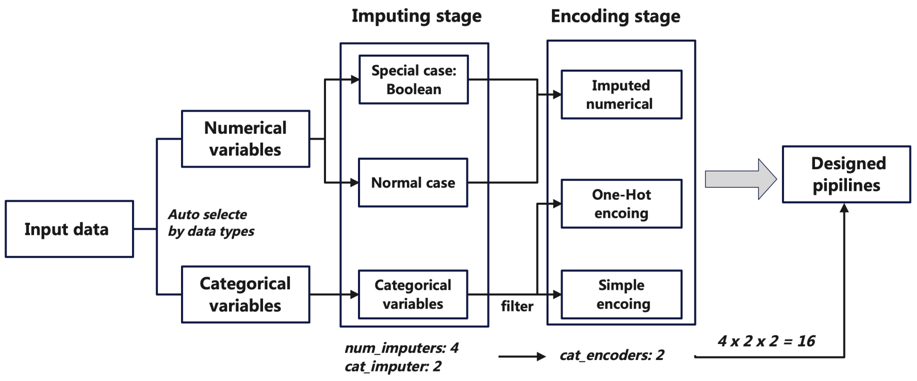


Figure 3: Design of Imputation Pipelines

Simple Encoding: For certain qualitative variables, simple imputation methods are not directly applicable due to their ordinal nature and the arbitrary encoding generated by default functions. To address this, we established a custom mapping scheme for these variables, as detailed in Table 3.

Table 3: Simple Encoding Mapping('–' indicates no mapping)

Encoded	Income	DNR	SFDMS (sfdm2)
0	Unknown	Unknown	–
1	Under \$11k	No DNR	No (Month 2 and SIP present)
2	\$11k-\$25k	DNR after SADM	ADL $\geq$ 4 (5 if survived)
3	\$25k-\$50k	DNR before SADM	SIP $\geq$ 30
4	\$50k	–	Coma or Intub
5	–	–	$<$ 2 mo. follow-up

One-Hot Encoding: Applied to multi-class categorical variables where the order of categories is not meaningful, such as `sex`, `dzgroup`, and `dzclass`.

4.2 Data Leakage Mitigation

To ensure model generalizability, we removed variables that could leak information about the outcome:

1) `death`, `hospdead`, `slos`, `d.time` — these variables represent outcomes or timing after the 2-month period and can leak post-prediction information into training.

2) Inclusion of these variables may indirectly reveal mortality and discharge timing, violating the principle that only pre-outcome data be used for prediction.

4.3 Stratified Train-Test Split

We performed an 80/20 split of the dataset, using stratified sampling to preserve the distribution of the `sfdm2` outcome variable(Figure 12 in Appendix). Standardization was applied to all features to reduce outlier influence and ensure consistent data scale.

5 Feature Selection

5.1 Multiple Logistic Regression

We applied MLR to each of selected sub datasets to identify the most accurate imputation strategy. The model predicts:

$$P(Y = 1 \mid X) = \frac{e^{\beta_0 + \sum_{j=1}^{p} \beta_j X_j}}{1 + e^{\beta_0 + \sum_{j=1}^{p} \beta_j X_j}}$$

where β_j are the coefficients estimated through log-likelihood maximization.

5.2 LASSO Regularization

We used LASSO regression to select the most important features from the dataset, and use the selected features to build the model. The optimization problem for LASSO regression can be formulated as:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^{\top} \beta \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

The reason we chose LASSO instead of ridge is that the LASSO penalty, $\lambda \sum_{j=1}^{p} |\beta_j|$, encourages sparsity in the coefficients, which drop some of coefficients to zero when some features are highly correlated(shown in Figure 1). However, ridge regression will not drop the highly correlated features, but will give them similar coefficients, which does not help to reduce the model complexity.

We tuned λ using a stratified 5-fold cross-validation over a log-transformed range: $\log(1/\lambda)$ from $[-4, 2]$.

5.3 Choosing of selected features and best imputation strategy

We evaluated model performance across the 16 imputation pipelines:

- **Best Imputation Strategy:** Imputer-2 achieved the highest accuracy of **71.87%**.
- **Optimal** $\lambda = 0.033598$, selected using 5-fold cross-validation.

The final LASSO model retained 23 predictors, balancing interpretability with predictive strength.

For model simplicity, we selected the 14 most important predictors based on their significance in the LASSO regression result and their contribution to predicting `sfdm2`.

6 Feature engineering for handling class imbalance

Class imbalance in classification tasks can be addressed partially through feature engineering techniques that modify the dataset to improve model performance on minority classes. Below, we explore three methods: Random Oversampling, SMOTE, and Augmented SMOTE.

- **Random Oversampling:**

  Random Oversampling duplicates minority class samples to balance class sizes with the majority class. While this method is computationally efficient, it may lead to overfitting due to repeated samples.

- **Synthetic Minority Oversampling Technique (SMOTE):**

  SMOTE generates synthetic samples for minority classes by interpolating between existing samples and their nearest neighbors. This approach ensures a more diverse dataset and reduces overfitting compared to simple oversampling.

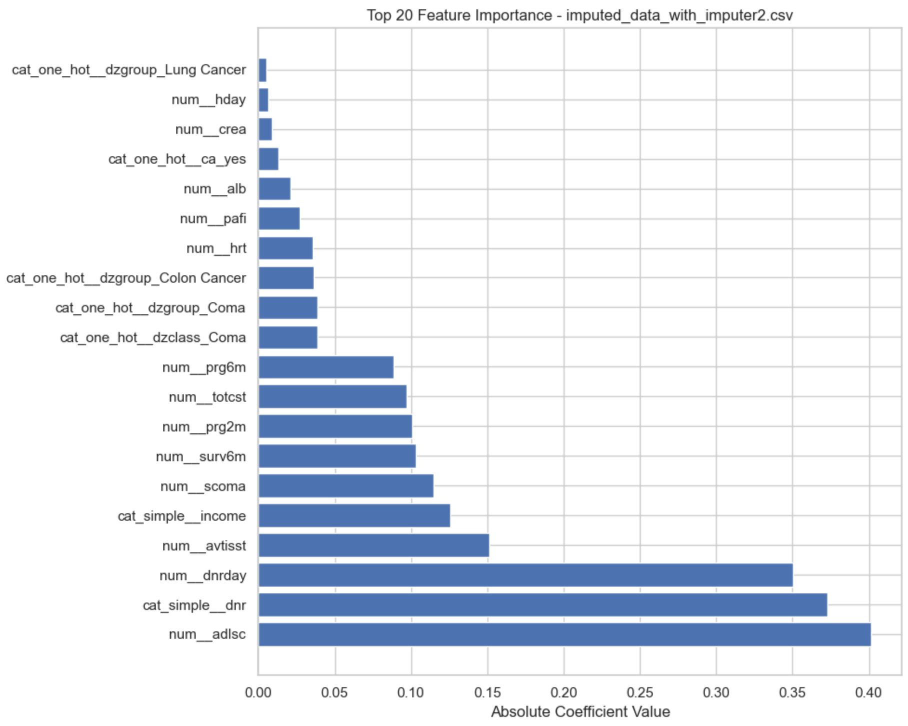


Figure 4: Rank of feature importantce in LASSO model

- **Augmented SMOTE with PCA:**

  Augmented SMOTE combines SMOTE with dimensionality reduction techniques like PCA to avoid over-synthesis in high-dimensional spaces. By reducing the data to a lower-dimensional representation before applying SMOTE.

  This method ensures meaningful interpolation and avoids overfitting. Additionally, random undersampling of majority classes can be applied to further balance the dataset.

6.1 Class Imbalance Mitigation

Exploratory data analysis revealed a strong class imbalance in the target variable (`sfdm2`), with most observations falling into Class 1 and Class 5, while Class 4 comprises less than 0.5% of the data. This skewed distribution can impair model performance on minority classes. To assess the impact, we trained a baseline Random Forest model on the original (imbalanced) data without any resampling or class weighting, and evaluated it on the test set. The performance results are summarized below.

Table 4: Classification Report (upon original Imbalanced Data)

Class	Precision	Recall	F1-score	Support
1	0.69	0.86	0.77	612
2	0.53	0.29	0.37	183
3	0.20	0.01	0.02	113
4	0.00	0.00	0.00	8
5	0.77	0.83	0.80	625
Accuracy	–		0.71	1541
Macro Avg	0.44	0.40	0.39	1541
Weighted Avg	0.66	0.71	0.67	1541

Although the overall accuracy reached 71%, the model failed to identify minority classes effectively. Class 4 was entirely missed, with precision, recall, and F1-scores of zero, and Class 3 was barely detected. This highlights the need to address class imbalance, as the model remains biased toward majority classes, limiting its fairness and applicability. To mitigate this, we explored three resampling strategies—Random Oversampling, SMOTE, and an augmented SMOTE variant—each aiming to enhance minority class performance through different mechanisms.

6.2 Random Oversampling method

Random Oversampling involves duplicating samples from minority classes to balance class distribution. In our case, we increased all class sizes to match the largest class (Class 5), which has 2,498 observations. The process is simple and fast, but may lead to overfitting due to repeated instances of the same samples.

Table 5: Classification Report (Random Oversampling)

Class	Precision	Recall	F1-score	Support
1	0.71	0.84	0.77	612
2	0.48	0.39	0.43	183
3	0.16	0.04	0.06	113
4	0.00	0.00	0.00	8
5	0.78	0.80	0.79	625
Accuracy	–		0.71	1541
Macro Avg	0.43	0.41	0.41	1541
Weighted Avg	0.67	0.71	0.68	1541

Compared to the baseline, performance for Classes 2 and 3 showed modest improvement. However, Class 4 remained undetected. While random oversampling helps mitigate class imbalance by increasing the representation of minority classes, it simply duplicates existing samples. This can lead to overfitting, as the model may memorize repeated instances rather than learning generalizable patterns. As a result, its effectiveness in improving performance on severely underrepresented classes remains limited.

6.3 Synthetic Minority Oversampling Technique (SMOTE) method

Unlike simple replication, SMOTE (Synthetic Minority Over-sampling Technique) generates synthetic examples by interpolating between existing minority class samples [2]. Specifically, for each minority instance, SMOTE identifies its k nearest neighbors in the feature space and generates new samples along the line segments connecting them [4]. Notably, SMOTE does not consider feature semantics or data distribution—it relies solely on geometric proximity, making it a domain-agnostic method for handling class imbalance. Since it depends on distance calculations, standardization of features is essential before application.

The simplified procedure of SMOTE involves: (1) identifying k nearest neighbors for each minority sample; (2) randomly selecting one or more of them; (3) generating new points by interpolation; and (4) repeating the process until the minority class reaches the target size.

In our study, we applied SMOTE to balance the dataset by increasing all class sizes to match the largest class (Class 5, with 2,498 samples). We then evaluated the same baseline model on this resampled data. Compared to both the original and random oversampling approaches, SMOTE yielded notable performance gains for several minority classes.

$$\text{new\_sample} = \text{original} + \lambda \cdot (\text{neighbor} - \text{original}), \quad \lambda \in (0, 1)$$

SMOTE improves minority class recall modestly but slightly reduces overall accuracy due to the introduction of synthetic samples. Compared to random oversampling, it's better at distributing attention to minority classes, though results still suggest further tuning or advanced methods.

Table 6: Classification Report (SMOTE)

Class	Precision	Recall	F1-score	Support
1	0.71	0.78	0.75	612
2	0.39	0.37	0.38	183
3	0.13	0.08	0.10	113
4	0.25	0.12	0.17	8
5	0.79	0.78	0.79	625
Accuracy	–		0.68	1541
Macro Avg	0.45	0.43	0.43	1541
Weighted Avg	0.66	0.68	0.67	1651

6.4 Augmented SMOTE with PCA

The original dataset was highly imbalanced, with Class 5 having 2,498 samples and Class 4 only 33. To reduce reliance on synthetic data, we specified custom target sizes per class rather than enforcing full balance. Random undersampling was applied to majority classes, while SMOTE was used for minority ones, preserving the relative class frequency structure.

To improve SMOTE's effectiveness in high-dimensional space, we applied PCA before neighbor selection and performed interpolation in the reduced space, then mapped synthetic samples back. This enhanced the quality and relevance of the generated data.

After tuning key parameters, we selected a PCA variance ratio of 90%, interpolation range $(0, 1)$, and class targets of [1000, 800, 700, 500, 1000]. This configuration balances class sizes while preserving feature variance and the natural class distribution.

Table 7: Classification Report (Augmented SMOTE)

Class	Precision	Recall	F1-score	Support
1	0.74	0.72	0.73	612
2	0.36	0.50	0.42	183
3	0.13	0.12	0.13	113
4	0.25	0.25	0.25	8
5	0.82	0.76	0.79	625
Accuracy	–		0.66	1541 samples
Macro Avg	0.46	0.47	0.46	1541
Weighted Avg	0.68	0.66	0.67	1541

After applying the adjusted training data into the same random forest baseline model, We achieved a test accuracy of 66%. The macro average F1-score improved to 0.46, indicating a better balance across all classes compared to baseline methods. Notably, performance for minority classes such as Class 2 (F1 = 0.42) and Class 3 (F1 = 0.13) showed modest gains, while Class 4, despite its extremely low support, achieved precision and recall of 0.25, suggesting a more robust representation in the model. Although the overall accuracy is comparable to traditional methods, augmented provides more balanced learning, helping the model recognize underrepresented classes while avoiding overfitting to synthetic examples.

7 Bayesian Improved Logistic Regression

In this section, we develop a Bayesian-enhanced logistic regression model using splines to capture potential non-linear relationships between predictors and the response variable. Instead of relying on classical Maximum Likelihood Estimation (MLE) to estimate model parameters, we adopt a Bayesian approach to infer their posterior distributions. Under this framework, the predicted probability of the target class (e.g., being 1) becomes a distribution that reflects parameter uncertainty. Final class predictions are made based on the expectation of the posterior distribution.

7.1 Accessible Module on GitHub

We implemented Bayesian logistic regression from scratch, allowing flexible extension in both model design and training. The full source code is available in our GitHub repository and can be accessed at:
`https://github.com/Gufeng-2002/5420_Final_Project/blob/main/src/LogsiticRegre.py`.

7.2 Transformation on design matrix for regression splines

Regression splines is a powerful tool to model the non-linear relationship between the input variables and the response variable. It splits the input space into several intervals and fits basis functions to each interval. The basis functions can be chosen to be simple linear or polynomial functions, or more complex functions. The functions across the intervals are connected at the knots, which makes sure the total regression function is continuous and smooth.

Given a univariate predictor $x \in R^n$, we can construct a regression spline design matrix by transforming x into a set of basis functions. For a spline of degree d with K knots $\{\xi_1, \xi_2, \ldots, \xi_K\}$, the new design matrix $\mathbf{X}_{\text{spline}}$ is:

$$\mathbb{K} : X(n,p) \mapsto X_{spline}(n, p+C), \quad C > 1$$

$$\mathbf{X}_{\text{spline}} = \begin{bmatrix} 1 & x_{11} & x_{11}^2 & x_{21} & x_{21}^2 & \cdots & x_{p1}^d & (x_{11}-\xi_{11})_+^d & \cdots & (x_{p1}-\xi_{pK})_+^d \\ 1 & x_{12} & x_{12}^2 & x_{22} & x_{22}^2 & \cdots & x_{p2}^d & (x_{12}-\xi_{11})_+^d & \cdots & (x_{p2}-\xi_{pK})_+^d \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 1 & x_{1n} & x_{1n}^2 & x_{2n} & x_{2n}^2 & \cdots & x_{pn}^d & (x_{1n}-\xi_{11})_+^d & \cdots & (x_{pn}-\xi_{pK})_+^d \end{bmatrix}$$

Here, $(x-\xi_j)_+^d$ denotes the truncated power basis function:

$$(x-\xi_j)_+^d = \begin{cases} (x-\xi_j)^d & \text{if } x > \xi_j \\ 0 & \text{otherwise} \end{cases}$$

This design matrix allows the regression model to fit a flexible, piecewise polynomial function with continuity at the specified knots.

7.3 The likelihood function of β in the logistic regression

To a data set having target variable that has only two vlaues, we view the target variable follows a Bernoulli distribution with true parameter p. The parameter is the probability of the target variable being 1 and is determined by the linear combination of the input variables X and the parameters β. It gives: $Y_i \sim Bernoulli(p_i)$.

By linking function - $log(\frac{x}{1-x})$, the conditional probability of the target variable Y given the input variables X is given by [1]:

$$log(\frac{p}{1-p}) = X\beta,$$

Where p is the vector of true predicted probabiltiy of Y being 1, X is design matrix of the input variables, and β is the vector of the parameters.

$$p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix}, \quad X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}.$$

[1] $\frac{p}{1-p}$ is doing broadcast operation, which makes sure the result is still a vector not linear algebra dot product.

After taking the inverse of the link function, we can get the predicted probability of the target variable being 1 [2]:

$$p = \sigma(X, \beta) = \frac{1}{1 + e^{-X\beta}}.$$

Based on the above probability model, we can get the likelihood function of parameters β given the data set (X, Y):

$$L(\beta|X, Y) = \prod_{i=1}^{n} p_i^{Y_i}(1 - p_i)^{1-Y_i} \tag{1-1}$$

Taking logarithm of the likelihood function and replacing p_i with $\sigma(X_i, \beta)$[3], we can get the log-likelihood function of the parameters β given the data set (X, Y):

$$\begin{aligned}\ell_n(\beta) &= \sum_{i=1}^{n}[y_i log(\sigma(x_i, \beta)) + (1 - y_i)log(1 - \sigma(x_i, \beta))] \\ &= \sum_{i=1}^{n}(x_i y_i \beta - log(1 + e^{x_i\beta}))\end{aligned}$$

We search for the maximum of the log-likelihood function to get the MLE of the parameters β:

$$\hat{\beta} = \arg\max_{\beta} \ell_n(\beta)$$

To find a good solution that gives a good likelihood function value, we can use the Newton-Raphson method to iteratively update the parameters β:

$$\beta_{k+1} = \beta_k - H^{-1}(\beta_k)\nabla\ell_n(\beta_k)$$

Where $H(\beta_k)$ is the Hessian matrix of the log-likelihood function $\ell_n(\beta)$ at β_k, and $\nabla\ell_n(\beta_k)$ is the gradient vector of the log-likelihood function $\ell_n(\beta)$ at β_k. The initial value of β can be set to any proper value, but a good choice can accelerate the convergence speed.

The Hessian matrix and the gradient vector can be calculated as follows [4]:

[2]'σ' is a sigmoid function, which takes X and β as input and returns the predicted probability, we will use it later for simplicity.

[3]X_i is the input vector of ith observation.

[4]We assume the $X's$ are independent that we can add n individual observed Fisher information to get the total Fisher information. H is summation of n matrices in (p, p) , $\nabla\ell_n(\beta)$ is the vector of the first derivative of the log-likelihood function

$$H(\beta) = -\sum_{i=1}^{n} \sigma(x_i, \beta)(1 - \sigma(x_i, \beta)) x_i x_i^T$$

$$\nabla \ell_n(\beta) = \sum_{i=1}^{n} (y_i - \sigma(x_i, \beta)) x_i$$

7.4 Prior about β from bootstrapping method

In situations where prior information about the regression coefficients $\boldsymbol{\beta}$ is unavailable or limited, a data-driven approach can be employed by using the bootstrap method to approximate the sampling distribution of $\boldsymbol{\beta}$. Specifically, we perform repeated sampling with replacement from the observed dataset and estimate $\boldsymbol{\beta}$ for each bootstrap replicate using a frequentist method such as ordinary least squares. This yields a collection of $\boldsymbol{\beta}$ estimates, from which we can compute the empirical mean $\hat{\boldsymbol{\beta}}$ and covariance matrix Σ_{boot}. These estimates can then be used to define an informative multivariate normal prior for Bayesian inference:

$$\boldsymbol{\beta} \sim \mathcal{N}(\hat{\boldsymbol{\beta}}, \Sigma_{\text{boot}})$$

This bootstrap-based prior reflects the variability observed in the data and provides a pragmatic empirical Bayes approach. While it does not represent a fully Bayesian treatment (as the prior is derived from the data), it enables regularization and uncertainty quantification in a principled way.

7.5 Bayesian improved logistic regression

After getting the likelihood function or β from original data and prior distribution of β from bootstrapping method, we can use the Bayes' theorem to get the posterior distribution of β:

$$\mathbb{P}(\beta | X, Y) = \frac{L(\beta | X, Y) p(\beta)}{p(X, Y)} \tag{1-2}$$

Where $p(X, Y)$ is the marginal likelihood of the data set (X, Y).

The posterior does not belong to any known distribution, but it is proportional to its numerator $p_{\beta_{post}} = \mathbb{P}(\boldsymbol{\beta} \mid \mathbf{y}, \mathbf{X})$:

$$\mathbb{P}(\boldsymbol{\beta} \mid \mathbf{y}, \mathbf{X}) \propto \ell_n(\beta) \cdot p(\beta)$$

it gives that we can get the 95% credible interval of β from this proportional unnormalized posterior distribution.

Based on the unnormalized distribution of β, we can get the posterior predictive distribution of the prability of target variable Y given the input variables X:

$$\mathbb{P}_{Y_{[\alpha,1-\alpha]}} = [X\beta_\alpha, X\beta_{1-\alpha}]$$

We set the probability of Y being 1 as the mean of the posterior predictive distribution of Y:

$$\mathbb{P}_{Y=1} = E[\beta_{post}X] = \frac{1}{B}\sum_{b=1}^{B}\sigma(X, \beta_b)$$

This modular design enables robust inference, handles nonlinearities via splines, and leverages empirical Bayesian estimation through bootstrapped priors.

7.6 One-VS-Rest classification

We use one-vs-rest method to transit from binary logistic regression to multi-class classification.

The logistic regression model for class k predicts the probability of an observation belonging to class k as:

$$P(y = k \mid \mathbf{x}) = \sigma(\mathbf{x}\boldsymbol{\beta}^{(k)}) = \frac{1}{1 + e^{-\mathbf{x}\boldsymbol{\beta}^{(k)}}},$$

where $\boldsymbol{\beta}^{(k)}$ is the parameter vector for class k.

After training all K models, the predicted class for a new observation $\mathbf{x}$ is determined by selecting the class with the highest predicted probability:

$$\hat{y} = \arg \max_{k\in\{1,2,\ldots,K\}} P(y = k \mid \mathbf{x}).$$

This approach allows binary logistic regression to handle multi-class problems by leveraging the simplicity of binary classification while maintaining interpretability and flexibility.

7.7 Implementation of deriving Prior Distribution of β

To make the modeling work simpler, we only use the data set processed by SMOTE method for all the work done in this section.

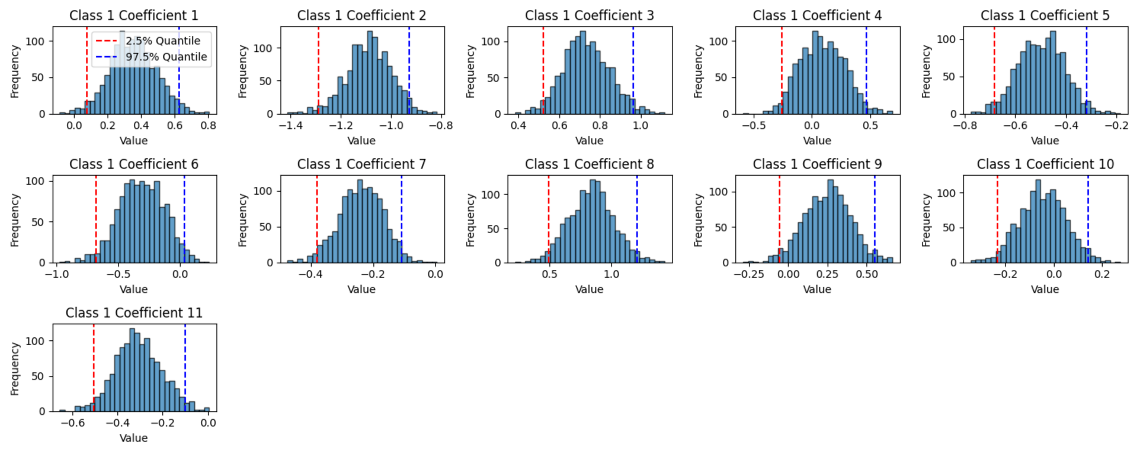


Figure 5: Bootstrap estimates histograms for prior distribution of β_i

Figure 5 [5] shows the empirical prior distributions of the logistic regression coefficients β_i estimated using bootstrap resampling for class label 1 among the five target categories. Each subplot corresponds to a coefficient in the model, including the intercept term, and design matrix($p = 11$) was not transformed in order to make computation faster.

The histograms represent the frequency distribution of coefficient estimates across bootstrap samples, giving an approximation of the sampling distribution under repeated sampling. Vertical dashed lines mark the 2.5% and 97.5% quantiles, providing a 95% empirical confidence interval for each β_i. These intervals reflect the uncertainty of the coefficients prior to incorporating the likelihood in the Bayesian inference step.

Figure 6 shows the histogram of values proportional to the posterior distribution of the logistic regression coefficients β. These values represent the product of the likelihood and the bootstrap-estimated prior, up to a normalization constant.

The vertical dashed lines indicate the 10% and 100% quantile thresholds, highlighting the top 90% of posterior-proportional values. This range identifies the most probable β candidates, useful for summarizing the posterior or computing the expected β [6].

7.8 Model performance evaluation

After repeatedly training the model on 5 different classes, we can get the model performance metrics of the models.

[5]Here we only use class 1 in response variable to demonstrate how we obtain the distribution of β_i from bootstrap method, that's there is 'Class 1' string in titles.

[6]The computation is also based on class 1 among the 5 classes.

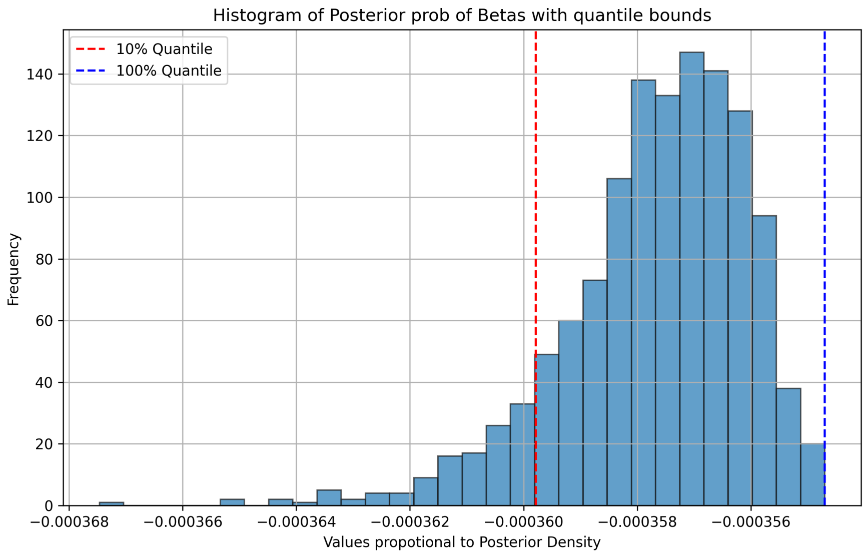


Figure 6: Hist of values proportional to the posterior distribution of β vector

Table 8 shows the model performance metrics of the performance of bayesian logsistic regression and MLE logistic regression model on training data.

Table 8: Model evaluation metrics grouped by target

Target	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Model**
1	0.8629	0.6111	0.8800	0.7213	MLE
	0.8629	0.6111	0.8800	0.7213	Bayesian
2	0.6774	0.3729	0.8800	0.5238	MLE
	0.7016	0.3889	0.8400	0.5316	Bayesian
3	0.6694	0.3095	0.5200	0.3881	MLE
	0.6855	0.3158	0.4800	0.3810	Bayesian
4	0.6048	0.2549	0.5417	0.3467	MLE
	0.6774	0.3095	0.5417	0.3939	Bayesian
5	0.7097	0.4000	0.8800	0.5500	MLE
	0.7419	0.4186	0.7200	0.5294	Bayesian

Table 8 compares the performance of MLE and Bayesian logistic regression models on the training dataset across five target classes using standard classification metrics.

Overall, both models exhibit consistent performance on class 1, achieving high accuracy (0.8629), precision (0.6111), recall (0.8800), and F1 score (0.7213), suggesting this class is well modeled.

For the remaining classes, Bayesian models tend to slightly outperform MLE models in terms of accuracy, precision, and F1 score. For example, in class 2, the Bayesian model achieves higher accuracy (0.7016 vs. 0.6774) and better F1 score (0.5316 vs. 0.5238), despite both models showing strong recall (above 0.84). This trend continues in classes 3 and 4, where the Bayesian model improves on accuracy and precision, although the recall values remain close. Class 4 shows a notable improvement in accuracy (0.6774 vs. 0.6048) and F1 score (0.3939 vs. 0.3467) for the Bayesian model.

In class 5, the Bayesian model again achieves higher accuracy (0.7419 vs. 0.7097), although its recall drops slightly compared to MLE (0.7200 vs. 0.8800), leading to a marginally lower F1 score. Overall, the Bayesian approach generally improves model robustness across imbalanced classes by balancing precision and recall more effectively, as reflected in consistently stronger F1 scores in most classes.

Table 9: Model evaluation metrics on test data

Target	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Model**
1	0.5887	0.1389	0.2000	0.1639	MLE
	0.5887	0.1389	0.2000	0.1639	Bayesian
2	0.4839	0.1695	0.4000	0.2381	MLE
	0.5081	0.1897	0.4400	0.2651	Bayesian
3	0.5403	0.1190	0.2000	0.1493	MLE
	0.5645	0.1081	0.1600	0.1290	Bayesian
4	0.5565	0.1961	0.4167	0.2667	MLE
	0.5968	0.1905	0.3333	0.2424	Bayesian
5	0.4516	0.1091	0.2400	0.1500	MLE
	0.5323	0.1333	0.2400	0.1714	Bayesian

The evaluation on test data reveals a noticeable drop in performance for both models across all metrics. Accuracy ranges from approximately 45% to 59%, and F1 scores remain low, mostly under 0.27. This indicates challenges in generalizing beyond the training data. Despite the overall low performance, the Bayesian model generally achieves slightly higher F1 scores than MLE, particularly for targets 2, 4, and 5, suggesting better balance between precision and recall under uncertainty.

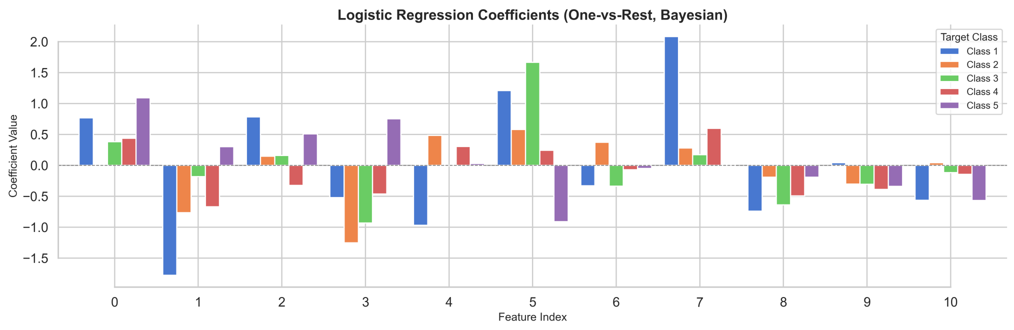


Figure 7: The importance of each feature in the Bayesian logistic regression model

Figure 7 illustrates the coefficients from five one-vs-rest Bayesian logistic regression models, each targeting a different class. Features with large absolute coefficient values have a stronger influence on the model's predictions. Notably, features indexed at 0, 1, 5, and 7 exhibit substantial variability across classes, indicating their high discriminative power. For instance, feature 6 is particularly important for Class 1 and Class 4, while feature 5 plays a dominant role in Class 3. Conversely, features 8–10 show relatively small coefficients across all classes, suggesting limited influence on prediction. The direction (sign) of each coefficient also reveals whether a feature increases or decreases the likelihood of the corresponding class.

Figure 8 displays the ROC curves for the five one-vs-rest Bayesian logistic regression models, each classifying one of the five target classes. The Area Under the Curve (AUC) serves as a summary of model performance, where higher values indicate better discrimination. Class 1 achieves the highest AUC (0.94), suggesting strong predictive performance. Class 5 and Class 2 also show good separation with AUCs of 0.85 and 0.82, respectively. In contrast, Class 3 (AUC = 0.70) and especially Class 4 (AUC = 0.67) demonstrate more limited classification ability, with curves closer to the diagonal, indicating greater confusion between classes. Overall, the ROC analysis highlights varying levels of model effectiveness across classes.

8 Non-parametric Classifiers

8.1 Hyperparameter Tuning

We used two widely known supervised learning algorithms, K-nearest neighbors (KNN) and Random Forest, to predict the levels of disability of patients. These models were selected because of their contrasting characteristics. KNN is a simple,

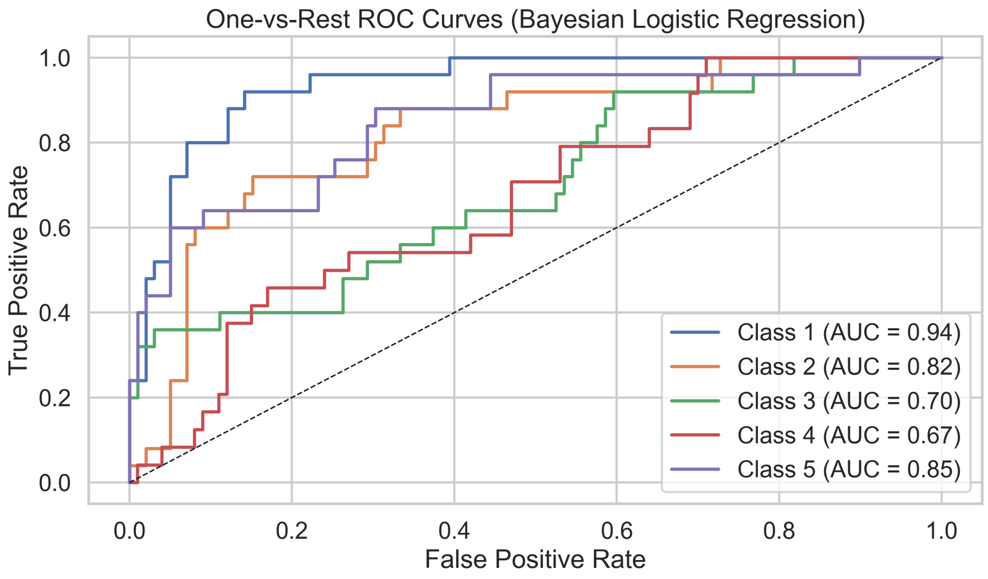


Figure 8: ROC curve of each of the Bayesian logistic regression models across 5 classes

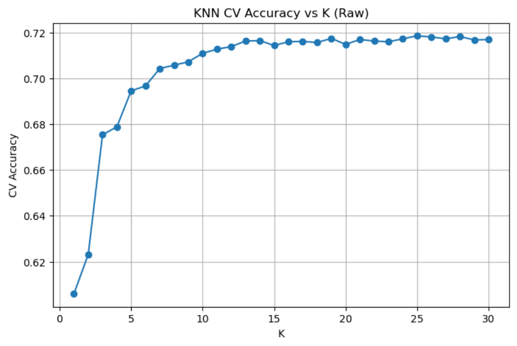
non-parametric classifier that makes predictions based on the majority label of its nearest neighbors, while Random Forest is an ensemble learning method known for its robustness, decorrelation between trees, and ability to handle high-dimensional datasets.

Like most machine learning models, both KNN and Random Forest rely on hyperparameters that can significantly influence predictive performance. Therefore, we applied k-fold cross-validation as the core evaluation method to select the best parameters before fitting the final models. This technique divides the training data into k folds, iteratively training the model on k–1 folds, and validating it on the remaining fold. This helps reduce overfitting and provides a more reliable estimate of model performance.

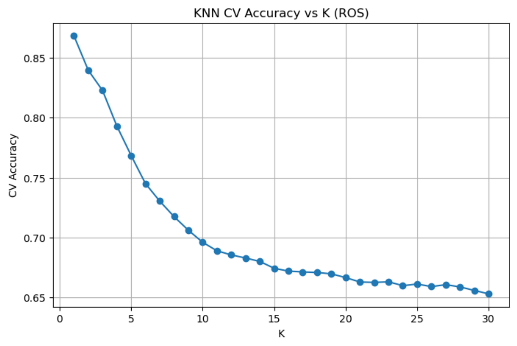
For the Random Forest algorithm, we additionally employed grid search to systematically explore multiple hyperparameter combinations. To ensure fair and consistent comparisons, this tuning process was repeated independently for each model on all four datasets (the raw, randomly oversampled, SMOTE-balanced, and advanced-balanced datasets).

8.1.1 K-nearest neighbors (KNN)

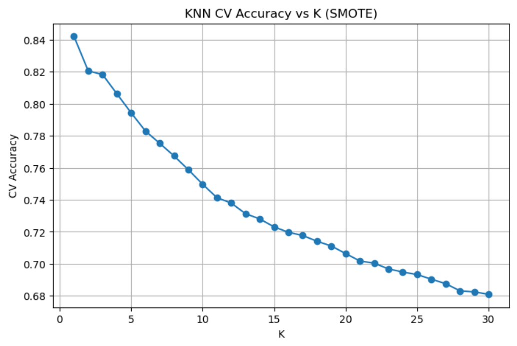
The most important hyperparameter for the KNN algorithm is k, which defines the number of nearest neighbors considered when making a prediction. To identify the optimal k value, we evaluated a range from 1 to 30 using cross-validation and selected the one that yielded the highest accuracy. As shown in the plots below, k = 25 was optimal for the original (imbalanced) dataset, while k = 1 performed best for the random oversampling, SMOTE-balanced, and advanced balancing datasets. This suggests that balancing the data may reduce the need to aggregate across neighbors, allowing the model to perform better with more localized patterns.



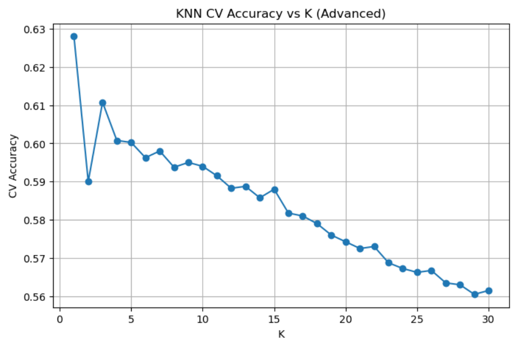
(a) KNN hyperparameter tuning for the original dataset.



(b) KNN hyperparameter tuning for the random oversampling dataset.



(c) KNN hyperparameter tuning for the SMOTE-balanced dataset.



(d) KNN hyperparameter tuning for the advanced-balanced dataset.

Figure 9: KNN hyperparameter tuning results for different datasets

8.1.2 Random Forest

In Random Forest, different hyperparameters need to be set and evaluated to achieve optimal model performance. Key hyperparameters include the number of trees in the forest (`n_estimators`), the number of features considered at each split (`max_features`), the maximum depth of each tree (`max_depth`), the maximum number of leaf nodes (`max_leaf_nodes`), and the minimum number of samples required at each node (`min_samples_leaf`). Given the numerous possible hyperparameter combinations, we performed a grid search over a predefined range to identify the optimal configuration based on cross-validation accuracy. Specifically:

- `n_estimators`: {50, 100, 200, 300, 500, 1000}
- `max_features`: from 1 to the total number of predictors
- `max_depth`: {5, 10, 15, 20, 30, Unlimited}
- `max_leaf_nodes`: {10, 20, 50, Unlimited}
- `min_samples_leaf`: {1, 2, 5, 10}

Based on cross-validation results, the optimal combinations of hyperparameters that achieved the highest validation accuracy are summarized in Table 10. These results suggest that different data balancing methods can influence the ideal model complexity and tree structure required for the best performance.

Dataset	`n_estimators`	`max_features`	`max_depth`	`max_leaf_nodes`	`min_samp_split`
Raw	100	6	10	50	1
Random Oversampling	300	4	30	Unlimited	1
SMOTE	300	4	30	Unlimited	1
Advanced Sampling	300	4	30	Unlimited	1

Table 10: Optimal hyperparameters for Random Forest across different datasets

8.2 Evaluation Metrics

Following the hyperparameter tuning process, we trained the KNN and Random Forest models on four versions of the dataset, including both imbalanced and resampled data. We applied several performance metrics, such as accuracy, sensitivity, and specificity, to evaluate the models. These metrics are commonly used to assess classification performance and are defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
$$\text{Sensitivity (Recall)} = \frac{TP}{TP + FN}$$
$$\text{Specificity} = \frac{TN}{TN + FP}$$

Where TP, TN, FP, and FN refer to true positives, true negatives, false positives, and false negatives, respectively.

In the following subsections, we present and discuss the results for each model across the four datasets. We begin with a comparison of overall accuracy, followed by detailed analyses of class-specific performance under each data condition.

8.3 Model Performance Across Datasets

8.3.1 Overall Accuracy Comparison

Before delving into class-wise metrics, Table 11 summarizes the overall accuracy of KNN and Random Forest across all datasets. Both models achieved the highest accuracy on the raw imbalanced data (KNN: 71%; RF: 72%), with gradual declines after applying resampling techniques. This observed trend suggests a trade-off between global accuracy and minority-class sensitivity. The subsequent sections analyze how resampling techniques enhanced sensitivity for minority classes, despite the reduced overall accuracy.

We now examine how each resampling strategy addresses this challenge, beginning with baseline performance on raw data.

Model	**Raw**	**Random Oversampling**	**SMOTE**	**Advanced Sampling**
KNN	0.71	0.63	0.60	0.55
Random Forest	0.72	0.71	0.69	0.67

Table 11: Overall Accuracy of KNN and Random Forest Across Different Sampling Methods

8.3.2 Raw Dataset

The results for predicting the levels of disability on the raw (imbalanced) dataset are presented in Table 12. Before applying any balancing techniques, both models achieved reasonable overall accuracy across all classes. They performed relatively well in terms of sensitivity and specificity for the more common categories, i.e., Classes 1 (mild or no disability) and 5 (death or loss to follow-up). However, both models showed very poor sensitivity for Classes 3 and 4, which represent severe levels of disability, including patients with serious quality of life decline (Class 3) and those in coma or tube insertion (Class 4). This is particularly concerning, as these classes are clinically critical and misclassifying such high-risk individuals could lead to serious consequences.

The poor sensitivity can be explained by how the algorithms function. As a lazy classifier, KNN relies heavily on the majority vote of the k nearest neighbors. When a class has very few samples, its data points tend to be more sparsely distributed and often surrounded by neighbors from majority classes. As a result, these minority classes have little influence during the voting process, which leads to poor sensitivity for Class 3 and 4 in this case. In the case of the Random Forest algorithm, using the Gini index as a splitting criterion makes the model focus on minimizing overall impurity. This causes it to prioritize the accuracy for majority classes, while minority classes are often ignored due to their small contribution to impurity. This explains why the Random Forest model also shows a sensitivity of 0 for Classes 3 and 4. To address these challenges and improve detection of underrepresented high-risk classes, we applied three different data balancing methods and re-evaluated the models accordingly.

Model	Class 1			Class 2			Class 3			Class 4			Class 5		
	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe
KNN	0.79	0.90	0.72	0.89	0.23	0.97	0.93	0.00	1.00	1.00	0.00	1.00	0.83	0.79	0.86
RF	0.80	0.88	0.75	0.89	0.32	0.97	0.93	0.00	1.00	0.99	0.00	1.00	0.84	0.84	0.84

Table 12: Performance Comparison on Raw Dataset
Note: Acc = Accuracy; Sen = Sensitivity; Spe = Specificity.

8.3.3 Random Oversampling

Table 13 compares the performance of the two models after training on a randomly oversampled version of the training set. While both models maintained relatively high accuracy and specificity across all classes, they continued to struggle with sensitivity for the minority classes (Class 3 and Class 4).

However, unlike on the raw dataset, KNN outperformed Random Forest in identifying these critical categories, achieving 19% sensitivity for Class 3 and 25% for Class 4, compared to just 4% and 0% for Random Forest. This suggests that random oversampling is more beneficial to KNN than to Random Forest. A likely reason is that random oversampling increases the presence of minority class samples in the training set, making them more likely to be included among the k nearest neighbors of a test instance. In contrast, although Random Forest is generally robust, it may overfit to repeated or noisy samples introduced during oversampling. As a result, it can struggle to generalize to unseen minority patterns in the test set. This performance gap highlights the need for more advanced balancing strategies to further improve sensitivity for underrepresented classes.

Model	Class 1			Class 2			Class 3			Class 4			Class 5		
	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe
KNN	0.75	0.71	0.77	0.83	0.28	0.91	0.88	0.19	0.94	0.99	0.25	0.99	0.80	0.73	0.84
RF	0.80	0.84	0.76	0.88	0.39	0.94	0.92	0.04	0.99	0.99	0.00	1.00	0.84	0.81	0.86

Table 13: Performance Comparison on Random Oversampling Dataset
Note: Acc = Accuracy; Sen = Sensitivity; Spe = Specificity.

8.3.4 SMOTE & Advanced Sampling

To further address class imbalance, we applied SMOTE and the advanced resampling method exclusively to the training set. The resulting model performances are shown in Tables 14 and 15. Compared to the results on the raw and randomly oversampled datasets, both techniques led to notable improvements in sensitivity for the minority classes (Classes 3 and 4). In particular, KNN showed enhanced performance, reaching 24–27% sensitivity on Class 3 and 25% on Class 4, outperforming Random Forest in these classes. On the other hand, Random Forest consistently maintained higher overall accuracy and specificity across most classes. Interestingly, while the advanced method provided the highest sensitivity for Class 3 among all models and methods tested, it came at the cost of reduced sensitivity for Class 1 and Class 5. This highlights a potential trade-off in model performance when prioritizing rare categories. These findings indicate that while the advanced balancing strategy can enhance sensitivity for underrepresented classes, it may also slightly impact predictions for majority classes.

Model	Class 1			Class 2			Class 3			Class 4			Class 5		
	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe
KNN	0.75	0.64	0.82	0.81	0.30	0.88	0.85	0.24	0.90	0.98	0.25	0.98	0.80	0.71	0.87
RF	0.80	0.79	0.80	0.86	0.39	0.92	0.89	0.09	0.96	0.99	0.12	1.00	0.84	0.79	0.87

Table 14: Performance Comparison on SMOTE Dataset
Note: Acc = Accuracy; Sen = Sensitivity; Spe = Specificity.

Model	Class 1			Class 2			Class 3			Class 4			Class 5		
	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe	Acc	Sen	Spe
KNN	0.72	0.54	0.84	0.78	0.36	0.84	0.82	0.27	0.86	0.97	0.25	0.98	0.79	0.65	0.88
RF	0.79	0.72	0.83	0.84	0.51	0.89	0.88	0.12	0.94	0.99	0.12	1.00	0.84	0.76	0.88

Table 15: Performance Comparison on Advanced Sampling Dataset
Note: Acc = Accuracy; Sen = Sensitivity; Spe = Specificity.

8.3.5 Summary of non-parametric models

Among all combinations, **KNN with SMOTE balancing** achieved relatively high sensitivity for the minority classes (Class 3: 24%; Class 4: 25%), while still maintaining competitive performance on the majority classes. Nevertheless, the relatively modest sensitivities still indicate room for further improvement.

9 Conclusion

In this project, we addressed the challenge of predicting functional disability levels (`sfdm2`) from the SUPPORT2 dataset, which consists of clinical and demographic data for critically ill patients. The prediction task involved a multi-class classification problem with significant class imbalance, particularly under-representation in Classes 3 and 4. To tackle this, we developed and compared multiple modeling approaches, including logistic regression (MLE and Bayesian), Random Forests, K-nearest neighbors (KNN), and several resampling strategies.

Initial exploratory data analysis revealed highly skewed class distributions and non-normal features, emphasizing the importance of robust preprocessing. We handled missing values via a combination of clinical rules and imputation pipelines, selecting the best strategy through model performance evaluation. Feature selection was performed using LASSO regression, allowing us to retain only the most relevant predictors and improve model interpretability.

For imbalance handling, we tested three resampling methods: Random Oversampling, SMOTE, and Augmented SMOTE with PCA. While simple oversampling led to marginal gains, SMOTE significantly improved recall for minority classes. The advanced SMOTE-PCA technique further enhanced performance by synthesizing data in a reduced-dimensional space, mitigating noise and overfitting risks.

Beyond classical MLE estimation, we implemented a Bayesian logistic regression framework with spline transformations to model non-linear relationships. Priors were derived from bootstrap estimates, enabling an empirical Bayes approach. The one-vs-rest strategy extended the binary logistic regression to multi-class classification. On one of the oversampling data sets, Bayesian models generally outperformed MLE models in minority class performance and provided more stable parameter estimates, albeit with slightly lower recall for some majority classes.

Performance evaluation on the test set showed all models struggled with generalization, particularly for rare classes. Nevertheless, Bayesian logistic regression demonstrated better balance in precision and recall, while Random Forests achieved the highest overall accuracy. KNN models, though less accurate, showed improved sensitivity to resampled data.

In summary, the combination of spline-based Bayesian modeling and advanced resampling offers a flexible and interpretable solution to class imbalance in clinical datasets. Our modular pipeline allows for robust estimation, informative priors, and principled multi-class extension. Future work may explore ensemble techniques, uncertainty quantification, and real-world deployment of models for clinical decision support.

References

[1] James, G., Witten, D., Hastie, T., Tibshirani, R., & Jankowski, H. (2023). *An Introduction to Statistical Learning: with Applications in Python.* Springer. https://www.statlearning.com/

[2] Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Journal of Artificial Intelligence Research*, *67*, 489–526. https://www.jair.org/index.php/jair/article/view/11192

[3] Beniwal, S. (2020). Data preprocessing steps for machine learning in Python (Part 1). *Medium.* https://medium.com/womenintechnology/data-preprocessing-steps-for-machine-learning-in-phyton-part-1-18009c6f1153

[4] He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *Journal of Artificial Intelligence Research*, *21*, 187–222. https://www.jair.org/index.php/jair/article/view/10302

[5] Mohammed, A. J. (2020). Improving Classification Performance for a Novel Imbalanced Medical Dataset using SMOTE Method. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(3), 3161–3172. https://doi.org/10.30534/ijatcse/2020/104932020

[6] Pan, T., Zhao, J., Wu, W., & Yang, J. (2020). Learning imbalanced datasets based on SMOTE and Gaussian distribution. *Information Sciences*, *512*, 1214–1233. https://doi.org/10.1016/j.ins.2019.10.048

[7] Swana, E. F., Doorsamy, W., & Bokoro, P. (2022). Tomek Link and SMOTE Approaches for Machine Fault Classification with an Imbalanced Dataset. *Sensors*, *22*(9), 3246. https://doi.org/10.3390/s22093246

[8] VanderPlas, J. (2022). *Python Data Science Handbook* (2nd ed.). O'Reilly Media. ISBN: 9781098121198.

[9] McKinney, W. (2017). *Python for Data Analysis* (2nd ed.). O'Reilly Media. ISBN: 9781491957660.

[10] Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and Statistical Modeling with Python. In *9th Python in Science Conference.*

[11] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (pp. 108–122).

10 Appendix

10.1 Figures

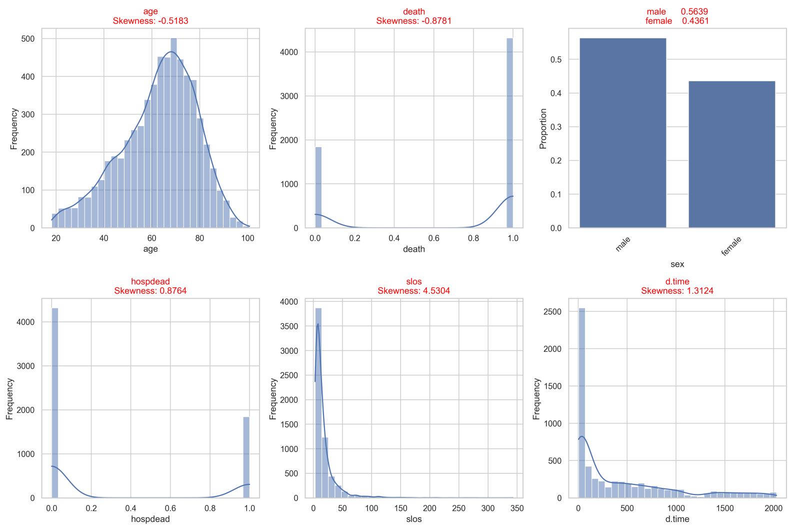


Figure 10: Descriptive statistics of the first 6 features SUPPORT2 dataset.

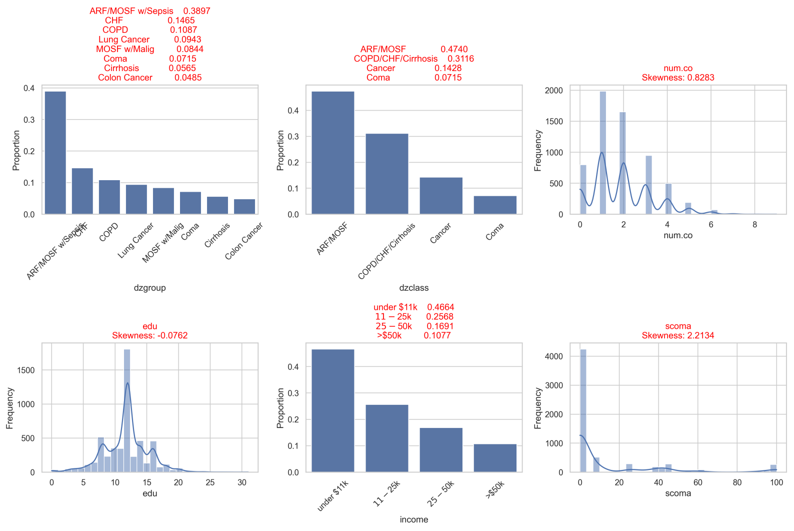
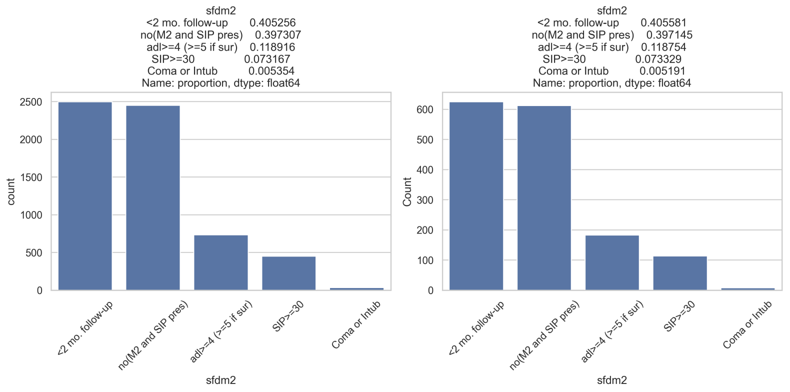


Figure 11: Descriptive statistics of the 7-12th features SUPPORT2 dataset.



```
sfdm2
<2 mo. follow-up       0.405256
no(M2 and SIP pres)    0.397307
adl>=4 (>=5 if sur)    0.118916
SIP>=30                0.073167
Coma or Intub          0.005354
Name: proportion, dtype: float64
```

```
sfdm2
<2 mo. follow-up       0.405581
no(M2 and SIP pres)    0.397145
adl>=4 (>=5 if sur)    0.118754
SIP>=30                0.073329
Coma or Intub          0.005191
Name: proportion, dtype: float64
```

Figure 12: Distribution of the target variable SFDM2 in train(left) and test(right) data sets

10.2 Tables

Table 16: Variable descriptions in the SUPPORT2 dataset

Variable Name	**Role**	**Type**	**Description**
id	ID	Integer	Patient ID
age	Feature	Continuous	Age of the patient in years
death	Target	Continuous	Death at any time up to NDI (Dec 31, 1994). Some patients are discharged before the end of study.
sex	Feature	Categorical	Gender (male, female)
hospdead	Target	Binary	Death in hospital
slos	Feature	Continuous	Days from Study Entry to Discharge
d.time	Feature	Continuous	Days of follow-up
dzgroup	Feature	Categorical	Disease subcategory: ARF/MOSF w/Sepsis, CHF, COPD, etc.
dzclass	Feature	Categorical	Disease category: ARF/MOSF, COPD, CHF/Cirrhosis, etc.
num.co	Feature	Continuous	Number of comorbidities (ordinal)
edu	Feature	Categorical	Years of education (missing values)
income	Feature	Categorical	Income level (missing values)
scoma	Feature	Continuous	SUPPORT day 3 Coma Score (Glasgow)
charges	Feature	Continuous	Hospital charges
totcst	Feature	Continuous	Total ratio of costs to charges (RCC)
totmcst	Feature	Continuous	Total micro cost
avtisst	Feature	Continuous	Average TISS score (days 3–25)
race	Feature	Categorical	Race: asian, black, hispanic, white, other (missing values)
sps	Feature	Continuous	SUPPORT physiology score on day 3
aps	Feature	Continuous	APACHE III day 3 physiology score
surv2m	Feature	Continuous	SUPPORT model 2-month survival estimate (day 3)
surv6m	Feature	Continuous	SUPPORT model 6-month survival estimate (day 3)
hday	Feature	Integer	Day in hospital at which patient entered study
diabetes	Feature	Continuous	Patient has diabetes (Yes/No)
dementia	Feature	Continuous	Patient has dementia (Yes/No)
ca	Feature	Categorical	Cancer status: yes, metastatic, no
prg2m	Feature	Continuous	Physician's 2-month survival estimate (missing values)
prg6m	Feature	Continuous	Physician's 6-month survival estimate (missing values)
dnr	Feature	Categorical	Do Not Resuscitate order status (missing values)
dnrday	Feature	Continuous	Day of DNR order (¡0 if before study)
meanbp	Feature	Continuous	Mean arterial blood pressure (day 3)
wblc	Feature	Continuous	White blood cell count (day 3)
hrt	Feature	Continuous	Heart rate (day 3)
resp	Feature	Continuous	Respiration rate (day 3)
temp	Feature	Continuous	Temperature (Celsius, day 3)
pafi	Feature	Continuous	PaO_2/FiO_2 ratio (day 3)
alb	Feature	Continuous	Serum albumin levels (day 3)
bili	Feature	Continuous	Bilirubin levels (day 3)
crea	Feature	Continuous	Serum creatinine levels (day 3)
sod	Feature	Continuous	Serum sodium concentration (day 3)
ph	Feature	Continuous	Arterial blood pH (day 3)
glucose	Feature	Integer	Glucose levels (day 3)
bun	Feature	Continuous	Blood urea nitrogen levels (day 3)
urine	Feature	Integer	Urine output (day 3)
adlp	Feature	Categorical	Index of ADL filled out by patient (day 3)
adls	Feature	Continuous	Index of ADL filled out by surrogate (day 3)
sfdm2	Target	Categorical	Functional disability level (1–5 scale, SIP questionnaire)
adlsc	Feature	Continuous	Imputed ADL Calibrated to Surrogate

Table 17: Variable descriptions in the SUPPORT2 dataset (summary statistics).

Variable	**Count**	**Mean**	**Std**	**Skewness**	**Kurtosis**	**Shapiro-p**
age	6164.00	62.85	15.72	-0.52	-0.15	0.00
death	6164.00	0.70	0.46	-0.88	-1.23	0.00
hospdead	6164.00	0.30	0.46	0.88	-1.23	0.00
slos	6164.00	17.38	20.89	4.53	33.64	0.00
d.time	6164.00	439.63	549.71	1.31	0.66	0.00
num.co	6164.00	1.89	1.36	0.83	0.65	0.00
edu	5330.00	11.72	3.44	-0.08	1.46	0.00
scoma	6163.00	13.07	25.88	2.21	4.13	0.00
charges	6046.00	58803.09	98338.95	4.58	30.48	0.00
totcst	5538.00	30783.65	45256.68	3.88	21.95	0.00
totmcst	3830.00	28725.87	43705.41	4.75	39.21	0.00
avtisst	6115.00	23.22	13.50	0.72	-0.17	0.00
sps	6163.00	26.04	10.08	1.58	5.31	0.00
aps	6163.00	38.71	20.33	0.92	0.92	0.00
surv2m	6163.00	0.62	0.26	-0.95	-0.10	0.00
surv6m	6163.00	0.51	0.26	-0.48	-0.90	0.00
hday	6164.00	4.55	9.43	6.07	57.29	0.00
diabetes	6164.00	0.20	0.40	1.49	0.21	0.00
dementia	6164.00	0.03	0.18	5.28	25.85	0.00
prg2m	5106.00	0.60	0.30	-0.54	-0.95	0.00
prg6m	5117.00	0.48	0.31	-0.13	-1.20	0.00
dnrday	6136.00	13.98	18.91	3.99	24.77	0.00
meanbp	6163.00	84.43	28.10	0.26	-0.25	0.00
wblc	6026.00	12.55	9.71	4.51	45.74	0.00
hrt	6163.00	97.66	32.05	0.19	0.59	0.00
resp	6163.00	23.52	9.69	0.48	1.05	0.00
temp	6163.00	37.10	1.26	0.32	-0.48	0.00
pafi	4680.00	237.86	109.34	0.89	1.35	0.00
alb	3905.00	2.94	0.91	6.43	174.96	0.00
bili	4437.00	2.66	5.55	4.64	26.23	0.00
crea	6123.00	1.78	1.67	3.12	13.10	0.00
sod	6163.00	137.57	6.06	0.35	1.33	0.00
ph	4708.00	7.41	0.08	-0.95	2.78	0.00
glucose	3153.00	159.72	88.02	2.44	10.87	0.00
bun	3238.00	32.25	26.59	1.98	6.40	0.00
urine	2886.00	2229.95	1485.34	0.96	1.22	0.00
adlp	2435.00	1.19	1.76	1.62	1.71	0.00
adls	4557.00	1.66	2.24	1.17	-0.04	0.00
adlsc	6164.00	1.86	2.05	0.97	-0.10	0.00

10.3 Bayesian Logistic Regression Algorithm Design Summary

This module implements a Bayesian logistic regression using spline-transformed features and bootstrap-based priors, organized under a one-vs-rest framework for multi-class classification.

- **Spline Feature Expansion:**

  ```
  transform_matrix_with_splines(X, degree=2, knots=quantiles)
      => returns expanded design matrix
  ```

- **MLE Estimation via Logistic Regression:**

  ```
  fit_logistic(X, y)
      => minimize negative log-likelihood:
         L($\beta$) = -mean[y log(p) + (1 - y) log(1 - p)]
  ```

- **Bootstrap-Based Priors:**

  ```
  Samples.bootstrap_estimates(X, y, B)
      => generates B bootstrap samples to estimate $\beta$ priors
  ```

- **Posterior Construction:**

  ```
  Posterior.posterior_estimates_density(X, y)
      => uses negative log-likelihood and prior samples
      => returns unnormalized posterior scores
  ```

- **Posterior Expectation of β:**

  ```
  posterior_quantile_betas(posterior, $\beta$, top_quantile=0.9)
      => computes mean $\beta$ from top quantile of posterior samples
  ```

- **One-vs-Rest Classification:**

```
fit_multiclass_logistic_ovr(data, num_classes=5)
    => fits class-specific $\beta$ for both MLE and Bayesian
```

- **Prediction:**

```
predict_multiclass(data, models)
    => computes predicted probabilities or labels per class
```