**Introduction: (Prepared by CHAN Shing-ho)**

Predicting mortality risk in the Intensive Care Unit (ICU) is important for several key reasons, including resource allocation, tailored interventions, informed decision-making, and the establishment of realistic care goals. In this project, we focus on three main components: data engineering, model analysis, and prediction. Each component employs various techniques to enhance prediction accuracy, which will be explained in detail below.

**Data Engineering: (Prepared by CHAN Shing-ho)**

1. Null Values: Our analysis revealed the presence of null values in our data. Since many machine learning (ML) algorithms cannot process null values, we will address this issue first by using K-Nearest Neighbours (KNN) imputation to modify the missing entries. The KNN imputation leverages the inherent relationships within the data and maintains the natural structure and distribution of the dataset. It can be applied to our data because all our features are numerical.
2. Feature Engineering: Creating new features ‘range’, ‘min to mean’, and ‘max to mean’ can enhance our dataset by providing additional insights and improving the performance of ML models. The range provides a simple measure of variability within a feature, indicating how spread out the values are. The ‘min to mean’ and ‘max to mean’ summarize two statistical properties into a single feature, which can simplify models while retaining important information. After creating new features, we expanded our dataset to a total of 66 features.
3. Z-score Normalization: During our analysis, we observed that the features have different units and scales. Therefore, we performed normalization to ensure that all features contribute equally to the analysis, preventing those with larger scales from dominating the model. Additionally, many machine learning algorithms, particularly those based on distance metrics, are sensitive to the scale of the data. Normalization enhances their performance by ensuring that distance calculations are meaningful.
4. Correlation Matrix: We examined the correlation matrix and the coefficients between the features and the target variable, 'mortality.' Unfortunately, we found that all coefficients are near zero. As a result, setting a threshold close to zero is ineffective for selecting strongly related features. Therefore, we will apply Principal Component Analysis (PCA) to identify combinations of features that explain the most variance in the data, even in the presence of low individual correlations.
5. A graph with a line

   Description automatically generatedPrincipal Component Analysis (PCA): PCA reduces the number of features while preserving the majority of the variance in the data. It identifies new axes that maximize variance within the dataset. Unlike methods that rely solely on linear correlations, PCA can uncover combinations of features that explain significant variance, potentially revealing underlying structures or relationships that may not be evident through correlation analysis alone. Ultimately, we selected 27 features for further analysis.
6. Outliers Identification: We utilized Isolation Forest to identify and remove outliers from our dataset. This machine learning algorithm employs a tree-based approach to detect outliers by randomly selecting features and splitting values; outliers are defined as points that can be isolated with fewer splits. We conducted parameter tuning on the Isolation Forest model to identify the optimal parameters. As a result, we identified 46 outliers. Additionally, we attempted to identify outliers using the Interquartile Range (IQR) method. While IQR is straightforward to understand and implement, it can be affected by skewed data, leading to potential misidentification of outliers. This resulted in the identification of 3,241 instances as outliers.
7. A graph of a blue and orange bar

   Description automatically generated with medium confidenceOversampling: After removing outliers, we further analysed the class distribution of our target variable. We discovered that the minority class '1' comprises only one-fourth of the dataset, indicating an imbalance. To address this issue, we applied the Synthetic Minority Over-sampling Technique (SMOTE) to achieve better balance. SMOTE generates synthetic samples for the minority class by interpolating between existing instances, rather than simply replicating them as in traditional oversampling. SMOTE considers the feature space of the minority class, creating new samples that are realistic and consistent with the distribution of the existing data. This process helps maintain the integrity of the dataset while enhancing the representation of the minority class.

**Modeling, Hyperparameter Finetuning, Model Interpretation: (By Wong Tsz Lun John Einstein 23472359)**

In the modelling part, we followed the Occam’s razor principle and started with a simple linear Latent Variable Model, in which, for each data point, we map the features **x** to a latent space **z** by a linear transform with an additional error term, i.e. z = **bT****x** + b0 + ɛ, where **w** is a vector of model parameters, w0 is the bias term, ɛ is an error term. We considered two different error distributions, namely 1) the standard normal distribution and 2) the standard logistic distribution. In the case of 1), the model is called **the (binary) probit model**, and for case 2), the model is called **the logistic model**, which we learned in lecture. The prediction is given by P(y=1| b, b0,x) = P(z≥0) = Φ(**bT****x** + b0) and P(y=0|**b**, b0,x) = P(z<0) = 1- Φ(**bT****x** + b0), where Φ(.) is the cumulative distribution of the chosen error distribution, and y⋳{0,1} is the binary output (mortality) we are predicting.

Since the dataset is class imbalanced, we added a weight hyperparameter w in the loss function to weight up the positive class and weight down the negative class to cancel out the effect of class imbalances. By inspecting the class distribution, we found that ~25% are positive class, we expect that w = 1 - 0.25 = 0.75, this is verified by GridSearchCV() implemented in cross\_validation.ipynb. We also added L1 and L2 regularization terms to the loss function, the objective function to be minimized is obj = -2\*(w\*y\*log Φ(**bT****x** + b0)+(1-w)\*(1-y)\*log Φ(-(**bT****x** + b0))) + l1\*|[**b** b0]| + l2\*|[**b** b0]|2 , where l1 and l2 are hyperparameters of L1 and L2 penalties, [**b** b0] denotes the concatenation of parameter vector and the bias term. (Note: Mathematically, we can omit the constant 2 in the loss, but we found it more numerically stable to include it, and we encode y to be {0,1} instead of {-1,1}, so the loss is different from the lecture notes). The loss is minimized using scipy.optimize.minimize, the implementation can be found in the classes probitModel(LatentVariableModel) and logisticModel(LatentVariableModel) in Models.py.

For hyperparameter finetuning, we separate the hyperparameter search of w and {l1, l2} independently since the former is dependent on class distribution, while the latter is data dependent. Apart from {l1,l2}, we also include n\_neighbors of KNN imputer together in the hyperparameter search. We applied Bayesian Optimization using the python package *bayesian-optimization* to search for the optimal hyperparameters.