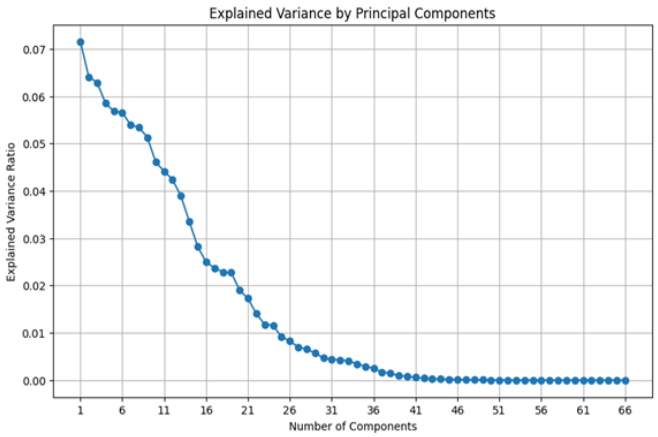
COMP7015 Group Project / Group 4 / Topic 1: ICU Mortality Prediction
Group Members: CHAN Shing-ho, HAN Hiu-wah, & WONG Tsz-lun John Einstein

Introduction: (By CHAN Shing-ho 23414537)

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 Pre-processing: (By CHAN Shing-ho 23414537)

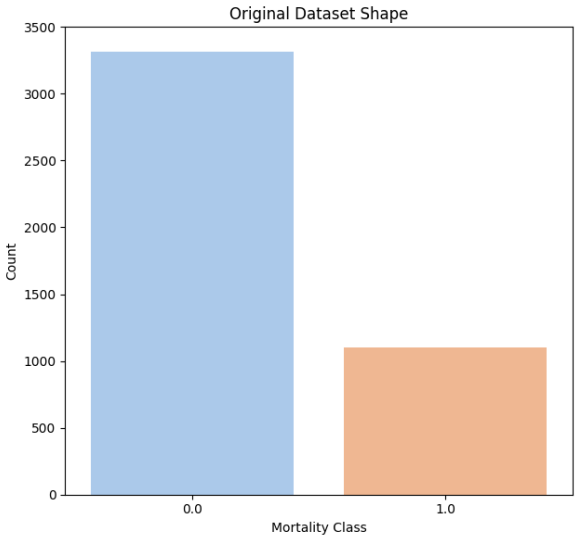
1. Null Values: Our analysis revealed the presence of null values in our data. Since many machine learning algorithms (MLA) cannot process null values, we will address this issue first by using K-Nearest Neighbours (KNN) imputation to modify the missing entries. The KNN imputation leverages the inherent relationships within the data and maintains the natural structure and distribution of the dataset.
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 MLA 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. Afterwards, we expanded our dataset to a total of 66 features.
3. Quantile Transformer (QT)[1]: During our analysis, we observed that the features have different units and scales. Therefore, we performed QT to ensure that all features follow a uniform or normal distribution. It can enhance the performance of certain algorithms that assume normally distributed data, such as linear regression or logistic regression. Our experiments showed that it yields better results than z-score normalization.
4. Principal Component Analysis (PCA): PCA reduces the number of features while preserving the majority of the variance in the data. Based on the results of the PCA, we typically will select around 27 features for further analysis. However, our experiments found that PCA would decrease our model's performance, therefore we did not add it to the data pre-processing pipeline at last. Details regarding the decline in performance will be presented in the model evaluation section.



5. Outliers Identification: We employed the Isolation Forest algorithm[2] to identify outliers in our dataset. This machine learning technique utilizes a tree-based approach to detect outliers by randomly selecting features and splitting values; outliers are defined as data points that can be isolated with fewer splits. We performed parameter tuning on the Isolation Forest model to determine the optimal settings, resulting in the identification of 46 outliers. However, the outlier predictions are correlated with our target variable, 'mortality', as confirmed by the Spearman rank test. Consequently, we incorporate the 'is_outlier' variable as a new feature. Additionally, we sought to identify outliers using the Interquartile Range (IQR) method. While the IQR method is easy to understand and implement, it can be influenced by skewed data, potentially

leading to the misidentification of outliers. This approach resulted in the identification of 3,241 instances as outliers, which are not useful for our project.

6. Oversampling: 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 examined the Synthetic Minority Over-sampling Technique (SMOTE)[3] 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 class balances of the dataset while enhancing the representation of the minority class. However, we later found that introducing class weights in the loss function can produce the same effect in an advantage of saving a step, hence processing time, in the pipeline. Therefore, we adopted the latter at last, which will be introduced in the next section.



Probit Model and Logistic Model, Hyperparameter Fine-tuning, Model Interpretation:
(By Wong Tsz Lun John Einstein 23472359)

Modeling : 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. $z = b^T x + b_0 + \varepsilon$, where w is a vector of model parameters, w_0 is the bias term, ε 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**[4], and for case 2), the model is called **the logistic model**[4], which we have learned in lecture. The prediction is given by $P(y=1| b, b_0,x) = P(z\geq 0) = F(b^T x + b_0)$ and $P(y=0|b, b_0,x) = P(z<0) = 1- F(b^T x + b_0)$, where F(.) is the cumulative distribution of the chosen error distribution, and $y\in\{0,1\}$ 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 sklearn's stratified GridSearchCV() implemented in hyperparameter_search.ipynb. We also added L1 and L2 regularization terms to the loss function, the objective function to be minimized is:

$$obj = -2(w \cdot y \cdot \log F(b^T + b_0) + (1 - w) \cdot (1 - y) \cdot \log F(-(b^T + b_0))) + l1 \cdot ||[b, b_0]|| + l2 \cdot ||[b, b_0]||^2$$

where l1 and l2 are hyperparameters of L1 and L2 penalties, [**b,** b_0] denotes the concatenation of the 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 encoded y to be {0,1} instead of {-1,1}, so the loss is different from the lecture notes). The loss was minimized using *scipy.optimize.minimize()* with *'SLSQP'* optimizer, the implementation can be found in the classes

probitModel(LatentVariableModel) and *logisticModel(LatentVariableModel)* in *Models.py*.

Hyperparameter Fine-tuning : 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 the hyperparameter *n_neighbors* of KNN imputers together in the hyperparameter search. We applied Bayesian Optimization (BO)[5] with stratified K-Fold cross validation (with K=5) using the python package *bayesian-optimization*[5] and *scikit-learn* to search for the optimal hyperparameters.

We choose stratified K-Fold because our class distribution is imbalanced, stratified K-Fold returns the stratified fold that preserve approximately the same class distribution in each stratum. BO is an optimization technique to maximize a black box function, in which we treat the cross validation as the black box function, and the hyperparameters as the input parameters to this black box function and the mean F1 score as the output of this function. The implementation is in hyperparameter_search.ipynb, the best hyperparameters are stored in **probit.json** and **logistic.json** respectively, and is shown in Table 4. The l1, l2 values we found are close to 0 for both models, it is expected since both are simple linear models, we expect underfitting instead of overfitting.

After searching for the best hyperparameters, we trained our models using all data, and performed a 5-fold cross validation to evaluate our models, F1 score of probit is slightly higher than that of logistic (0.4778 vs 0.4764), suggesting that the gaussian error assumption may fit the data better. All the F1 score calculations in our experiments are using 0.5 as the prediction threshold. The source code is in train_probit_logistic.ipynb, and the ROC curve, confusion matrices, and the evaluation result of each fold and overall statistic (**result.txt**) can be found in the folder ***probit*** and ***logistic*** respectively, a comparison with LightGBM model can also be found in Table 4 in the next section.

Model Interpretation: We can interpret the feature importance using the Wald test[7]. The square rooted Wald statistic measures the distance between the estimated parameters and 0, normalized by standard error (S.E.) of the estimate:$\sqrt{W} = |\frac{\theta}{S.E.}|$, where θ is [b, b0]. The p-value is given by $p = 2(1 - \Phi(|\frac{\theta}{S.E.}|))$, where $\Phi(.)$ is the standard normal distribution. The implementation can be found in the *wald_test()* function in the *LatentVariableModel* class.

To interpret the model, a low p-value indicates that the corresponding feature significantly affects the predicted outcomes. If the fitted coefficient is positive, it means higher feature value increases the chance of mortality, while negative coefficient means higher feature value decreases the chance of mortality. We selected the top 5 significant features by sorting p-value in ascending order for discussion. The top 5 significant features found by both models are the same, except with different orders. The result can be found in train_probit_logistic.ipynb, and is shown below:

Table 1: Top 5 features of probit model

feature	coef	std err	z	P>|z|
Oxygen saturation_min	-0.048209	0.0329	-1.4637	0.1433
Respiratory rate_mean	0.095568	0.0862	1.1089	0.2675
Temperature_min	-0.056358	0.0521	-1.0824	0.2791
is_outlier	-0.057193	0.0577	-0.9913	0.3215
Temperature_mean	-0.068612	0.0791	-0.8672	0.3858

Table 2: Top 5 features of logistic model

feature	coef	std err	z	P>|z|
Oxygen saturation_min	-0.105109	0.0551	-1.9064	0.0566
is_outlier	-0.151197	0.0978	-1.5462	0.1221
Respiratory rate_mean	0.182200	0.1419	1.2836	0.1993
Temperature_mean	-0.145546	0.1300	-1.1199	0.2627
Temperature_min	-0.093339	0.0847	-1.1015	0.2707

The explanations for the results are:

- Higher Oxygen saturation_min decrease mortality risk *(oxygen is vital for life)*
- Higher Respiratory rate_mean increases mortality risk
  (high respiratory rate means the body is starving for oxygen)
- Higher Temperature_min decrease mortality risk
  (low body temperature (hypothermia) means weaker vital sign)
- Inliniers (1 for inliner, -1 for outlier) lower mortality risk
  (outliers correlates with mortality risk, since mortality is the minority class in the data)
- Higher Temperature_mean decrease mortality risk *(same explanation as Temperature_min)*

LightGBM (By Chan Hiu-wah)

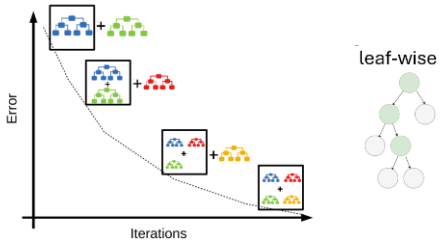
1. Objective

LightGBM[8] is a fast and memory efficient gradient boosting tree model. It is a leaf-wise tree model and efficiently splits the leaves with the largest gradient loss. It creates a complex tree structure efficiently and may easily get overfitted in small amounts of data.

2. Model Training

Binary log loss for binary classification {0,1}[9]

$$Log\ loss = -\frac{1}{N}\sum_{i=1}^{N}(w_i y_i \log(y_i) + w_i(1-y_i)\log(1-y_i))$$



- Weight (w) is the adjustment to the loss function by the ratio of imbalance class.. It gives more weights (importances) to the unbalanced data while training the model.
- Gradient Boosting method with D.A.R.T (Dropouts meet Multiple Additive Regression Trees)[10]. It regulates the model by randomly dropping out some trees to construct new trees.

3. Hyperparameter fine-tuning

Bayesian optimization for hyperparameter tuning: it uses Bayesian approaches P(Score|hyperparameter) to construct a probabilistic model to optimize the parameters tuning.

Table 3: Hyperparameters of the LightGBM model

n_neighbors: number of neighbors used in data preprocessing for handling null values using KNN algorithm, value: 16	bagging_fraction: portion of data to be used in bagging, enhance robustness and reduce variance value: 0.5935166476056839
lambda_l1: L1 regulation (feature selection) value: 4.453297733821403	lambda_l2: L2 regulation value: 4.633817416757843
max_bin: number of bins used to categorize features values, value: 67	feature_fraction: portion of features used in each boosting iteration, value: 0.5485892932500758
drop_rate: used in DART, portion of trees dropping out in each iteration, value: 0.3622062793963785	max_drop: used in DART, maximum number of dropped trees in each iteration, value: 27
max_depth: maximum depth of tree, avoid overfitting, value: 3	num_leaves: max number of leaves in tree, avoid overfitting, value: 71

min_split_gain: minimum gain after splitting, avoid overfitting, value: 0.04815891792473386	min_data_in_leaf: minimum number of data in each leaf, avoid overfitting, value: 21

* Top 5 Feature importance: Respiratory_rate_mean (420), Temperature_mean(349), Systolic_blood pressure_min(334), Glucose_mean(315), PH_mean(301)

Model prediction result using Stratified K-Fold validation (By Chan Hiu-wah)

Instead of random sampling, stratified K-Fold cross validation splits data in the same proportion according to their class label. It reduces the model variances due to imbalance class data.

Probit model and logistic model have similar scores, while probit model has highest recall value. In practice, it correctly identifies the predicted mortality of risky patients and can allocate treatment to the risky patients. LightGBM model has the highest precision, which means that LightGBM model is less likely to predict incorrectly and allocate excessive resources to non-risky patients. LightGBM model has the highest AUC (model performance) and F1 score (harmonic mean of precision and recall).

Table 4: Comparison of the prediction results of different models

	Probit Model	**Logistic Model**	**LightGBM Model**
Hyperparameters	l1= 0.00617, l2= 0.04057, n_neighbors=11	l1=0.00274, l2= 0.00904, n_neighbors=14	l1= 4.45330,l2=4.63382, n_neighbors=16
F1 score	0.47780 ± 0.01337	0.47644 ± 0.01573	**0.51456 ± 0.00686**
AUC	0.68860 ± 0.02094	0.69117 ± 0.01810	**0.74691 ± 0.00937**
Precision	0.37777 ± 0.01301	0.38676 ± 0.01482	**0.44828 ± 0.00754**
Recall	**0.65018 ± 0.01499**	0.628268 ± 0.02206	0.60424 ± 0.01596
Accuracy	0.64370 ± 0.01357	0.656804 ± 0.01481	**0.71436 ± 0.00615**

Group distribution

CHAN Shing-ho 23414537	WONG Tsz-lun John Einstein 23472359	Chan Hiu Wah 23467509
Data Pre-processing	Probit and logistic model, hyperparameter finetuning of these 2 models	Modeling, parameter tuning, cross validation, testing and evaluating LightGBM model, including programming and report parts

Reference

1. “sklearn.preprocessing.QuantileTransformer,” scikit-learn, [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.QuantileTransformer.html. [Accessed: Nov. 19, 2024].
2. “sklearn.ensemble.IsolationForest,” scikit-learn, [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.IsolationForest.html. [Accessed: Nov. 19, 2024].
3. “imblearn.over_sampling.SMOTE,” imbalanced-learn, [Online]. Available: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html. [Accessed: Nov. 19, 2024].
4. J. L. Horowitz and N. E. Savin, “Binary Response Models: Logits, Probits and Semiparametrics,” Journal of Economic Perspectives, vol. 15, no. 4, pp. 43–56, Dec. 2001, doi: 10.1257/jep.15.4.43.
5. Fernando Nogueira. (2014–). Bayesian Optimization: Open source constrained global optimization tool for Python.
6. “sklearn.model_selection.StratifiedKFold,” scikit-learn, [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.StratifiedKFold.html. [Accessed: Nov. 19, 2024].
7. Davidson, R. and J. G. MacKinnon. Econometric Theory and Methods. Oxford, UK: Oxford University Press, 2004.
8. G. Ke et al., “LightGBM: a highly efficient gradient boosting decision tree,” in Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 3149–3157.
9. M. Filho, “LightGBM for Binary Classification in Python,” Forecastegy, Nov. 14, 2023. [Online]. Available: https://forecastegy.com/posts/lightgbm-binary-classification-python/. [Accessed: Nov. 19, 2024].
10. K. V. Rashmi and R. Gilad-Bachrach, DART: Dropouts meet Multiple Additive Regression Trees. 2015.