HW2_Report

Part 1:

The language I used was Python, and the libraries for data preprocesing I imported were as below:

```
# Essential DS libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from scipy import stats
```

1. Data Pre-processing(data_preprocessing.py)

a. Dataset analysis:

i. Overview of Dataset:

1. train_X.csv: The dataset included various patient characteristics that served as input data for the model.
2. train_y.csv: The dataset included the target variable ”has_died” which indicated whether patients have died. The value “0” means “alive”, the value “1” means “die”.
3. test_X.csv: The test dataset was used to generate the final prediction.
4. sample_submission.csv: It was the template before uploading to Kaggle.

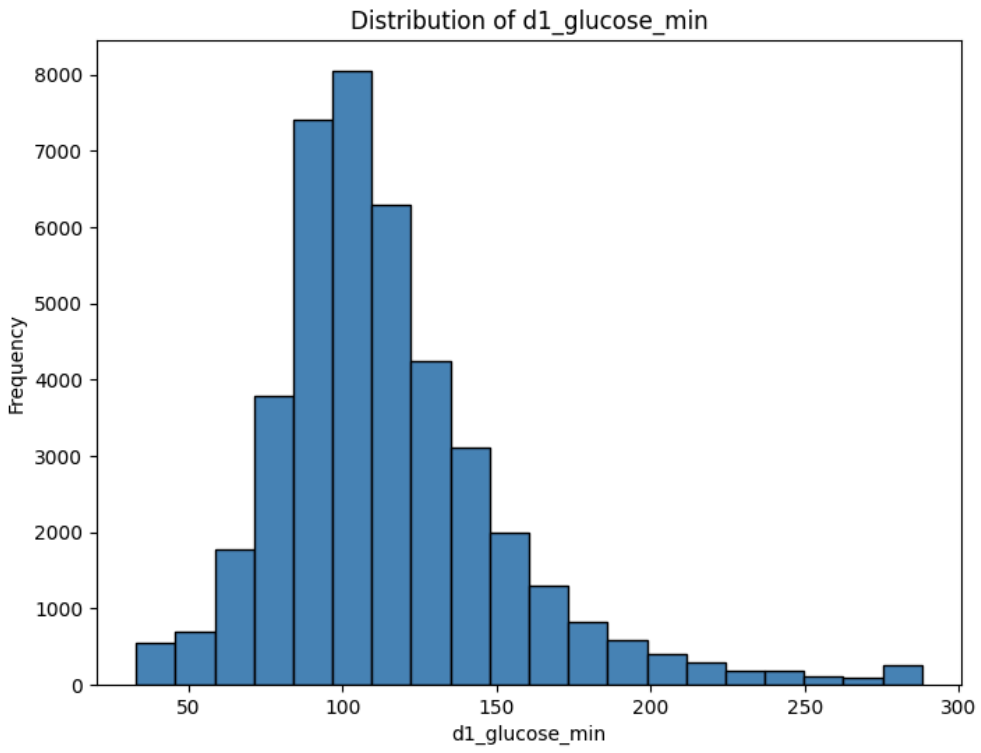
ii. Data Statistics:

1. I applied “list(train_X.select_dtypes(include='object').columns)” and “list(train_X.select_dtypes(include=['int64', 'float64']).columns)” to extract the categorical and numerical features from the train_X dataset. However, I noticed that some of the numerical features should be classified as categorical. Therefore, I used “[col for col in numeric_features if train_X[col].nunique() <= 6]” to identify columns with unique values ranging from 0 to 6, appended them to the categorical features list, and removed them from the numerical features list.
2. I used “train_X.info()” function to view the basic structure of data, then I can know more about data types and missing values. In “train_data”, there were three different data types such as float64, int64, object and most of features exist missing values.
3. I used “train_X.describe()” function to get statistical information of numerical features such as count, mean, std, min, max, median, etc.

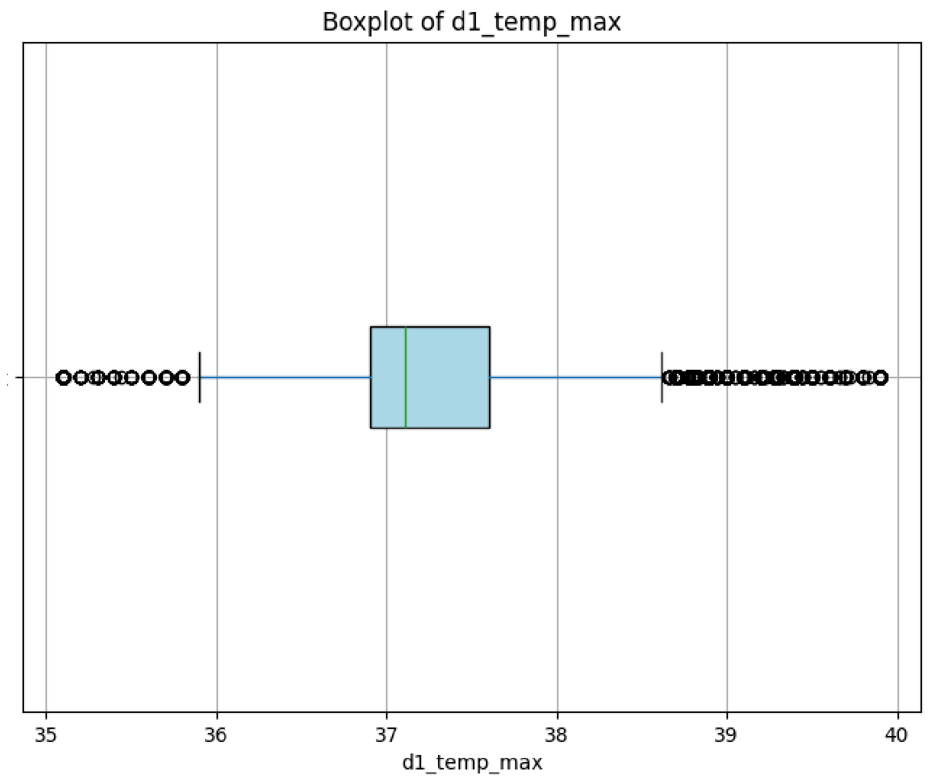
4. I used “value_counts()” method to know more about frequency distribution of categorical features.

iii. Data Visualization:

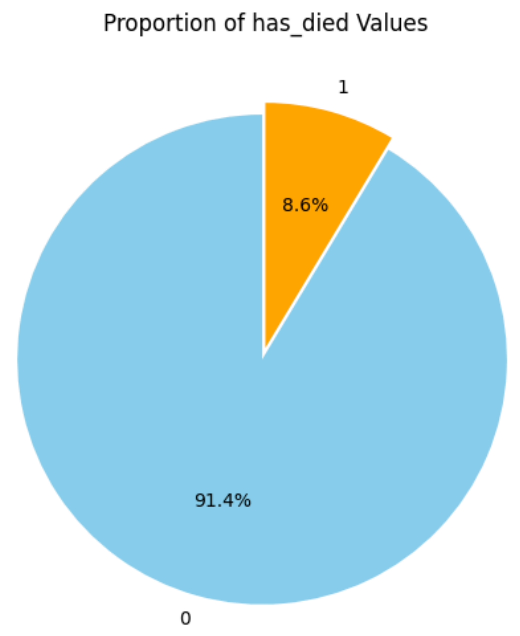
1. numerical features:
   a. In “train_X”, there were 59 numerical features such as heart_rate_apache, d1_mbp_max, d1_mbp_noninvasive_min, etc.
   b. histogram: In order to understand the distribution of each numerical feature, I used a for loop method to scan through each numerical feature and drew the corresponding histogram. Here is the histogram of d1_glucose_min.



   c. boxplot: In order to understand the maximum value, minimum value, median, upper and lower quartiles of each numerical feature, and whether there are outliers, I used boxplots to draw charts. I observed that there were many outliers for “patients vital signs” such as pre_icu_los_days, temp_apache, d1_diasbp_max, d1_diasbp_min, d1_diasbp_noninvasive_max, d1_diasbp_noninvasive_min, d1_heartrate_max, d1_heartrate_min, d1_mbp_max, d1_mbp_min, d1_mbp_noninvasive_max, d1_resprate_max, d1_resprate_min, d1_spo2_max, d1_spo2_min, d1_sysbp_max, d1_sysbp_min, d1_sysbp_noninvasive_max, d1_sysbp_noninvasive_min, d1_temp_max, d1_temp_min, etc. Here is the boxplot of d1_temp_max.



2. categorical features:
    a. In “train_X”, there were 24 categorical features such as ethnicity, gender, icu_admit_source, icu_stay_type, icu_type, apache_3j_bodysystem, apache_2_bodysystem.
    b. bar plot: Using bar plot to understand frequency of each categorical feature.
    c. pie chart: In “train_y”, it was observed that the proportions of "has_died" values for 0 and 1 differ significantly. I used a pie chart to visualize the proportions, which shows that the proportion of non-deceased cases accounts for 91.4%, while deceased cases account for 8.6%. Pie chart is as below.



b. Data cleaning:
   i. Delete row data with a missing value ratio exceeding 50%:
      1. In order to delete row data with a missing value ratio exceeding 50%, I first calculated the total number of missing values in each row, then calculated its ratio, and then added two columns "Missing Count" & "Missing Ratio" to "train_X". And fill in the total number and proportion. I set the threshold to 0.5. When Missing Ratio > threshold, the row is dropped. After calculation, a total of 60 pieces of data need to be deleted, so the number of rows in the cleaned "train_X" and “train_y” becomes 44879.
      2. The “train_X” dataset renamed as “train_X_cleaned”.
      3. The “train_y” dataset renamed as “train_y_cleaned”.
c. Data imputation:
   i. Fill missing values:
      1. For numerical features, I applied SimpleImputer(strategy='median') function which is a median imputation method to handle missing values.
      2. For categorical features, I viewed missing values as new categories, and used “fillna(“Missing”)” method to replace missing values with “Missing”.
      3. The “train_X” dataset renamed as “train_X_imputed”.
d. Data transformation:
   i. Encoding Categorical Features with LabelEncoder():
      1. Because SMOTE requires numerical input features and cannot directly process categorical data. To address this issue, all categorical features in the “train_X_imputed” dataset were transformed into numerical

representations using Label Encoding. This encoding assigns unique integer values to each category within a feature.

e. Data imbalance handling:

   i. Brief explanation of Synthetic Minority Over-sampling Technique:

      1. The “train_y” dataset exhibits a significant imbalance in the target variable “has_died”, where the number of minority class samples (e.g., patients who died) is substantially lower than that of the majority class. This imbalance can lead to biased model predictions, as the model tends to favor the majority class. To address this issue, I applied SMOTE to balance the dataset by oversampling the minority class. SMOTE works by generating synthetic samples between existing minority class samples to balance the class distribution. Unlike simple duplication, these synthetic samples are created through linear interpolation of feature values among neighboring samples in the minority class. This approach preserves data diversity while reducing the risk of overfitting.

   ii. Oversampling with SMOTE:

      1. To address the imbalance in the has_died target variable, where the minority class is underrepresented. I applied SMOTE to generate synthetic samples for the minority class by using linear interpolating between existing minority class samples. The sampling_strategy(=0.2) parameter set the minority class sample to 20% of the majority class. After SMOTE, the “dataset achieves a more balanced class distribution.

2. Classification Methods(train_classification_model.py)

The libraries for model training I imported were as below:

```
# Standard python libraries
import os
import time
import requests
# Essential DS libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.metrics import roc_auc_score, f1_score
from sklearn.model_selection import KFold
import torch
# LightAutoML presets, task and report generation
```

```
from lightautoml.automl.presets.tabular_presets import TabularUtilizedAutoML
from lightautoml.tasks import Task
import joblib
```

a. Describe and introduce clearly the machine learning algorithms:
    i. Objective:
    This report provides a detailed explanation of the Python script train_classification_model.py, which trains a binary classification model using the LightAutoML framework. The model aims to predict patient mortality (has_died) based on clinical features. The process includes data preprocessing, hyperparameter tuning, model training with KFold cross-validation, and performance evaluation using metrics such as AUROC and F1-score.
    ii. Overview of LightAutoML:
    LightAutoML is an open-source automated machine learning framework designed for efficient and interpretable modeling on tabular datasets. It employs state-of-the-art algorithms, hyperparameter optimization, and advanced model ensembling techniques to deliver robust results.In this experiment, LightAutoML utilized the following machine learning algorithms:
        1. LinearL2 Regression:
            a. LinearL2 Regression, also known as Ridge Regression, is a linear regression model that incorporates L2 regularization. This regularization penalizes large coefficients, which helps to prevent overfitting, particularly when the dataset contains multicollinearity (highly correlated features).
            b. Reference: E. Hoerl and Robert W. Kennard (1970), "Ridge Regression: Biased Estimation for Nonorthogonal Problems."
        2. LightGBM:
            a. LightGBM is an open-source gradient boosting framework that is optimized for performance and scalability. Using Decision Trees as base learners, LightGBM builds trees sequentially, where each new tree corrects the errors of the previous ones by focusing on misclassified samples. Instead of row sampling, it optimizes the leaf-wise growth of decision trees. It is widely used for tabular data tasks due to its speed and accuracy.
            b. Reference: Ke et al. (2017), "LightGBM: A Highly Efficient Gradient Boosting Decision Tree."
            c. Packages: lightgbm
        3. CatBoost:

        a. CatBoost is a gradient boosting framework specifically optimized for datasets with categorical features. Instead of training on all data, CatBoost trains each model on a subset and uses the remaining data to calculate target statistics. It provides state-of-the-art performance while requiring minimal preprocessing.
        b. Reference: Prokhorenkova et al. (2018), "CatBoost: Unbiased Boosting with Categorical Features."
        c. Packages: catboost
    4. Model Ensembling:
        a. LightAutoML employs a stacking strategy to combine multiple base models (LinearL2, LightGBM, and CatBoost) into a single ensemble model. This approach helps improve generalization by leveraging the strengths of different algorithms.
        b. Reference: D.H. Wolpert (1992), "Stacked Generalization."

iii. Parameters Applied and Tuning:
    1. General Parameters:
        a. Number of threads: Specifies the number of CPU threads allocated for model training. Parameter: N_THREADS = 12
        b. Number of folds: The number of folds for KFold cross-validation. Parameter: N_FOLDS = 5
        c. Timeout: Specifies the total time allocated for training the model. Parameter: TIMEOUT = 7200 seconds (2 hours)
        d. Target_name: Specifies the target variable for the classification task (has_died). Parameter: TARGET_NAME = 'has_died'

iv. The whole splitting process of KFold cross validation:
    1. Data Splitting: The “train_data” dataset is split into N_FOLDS equal parts.
    2. Fold Iteration: Each fold is used as a validation set once, while the remaining folds form the training set.
    3. Model Training and Evaluation: The model is trained and evaluated in each fold, and performance metrics (AUROC and macro F1-score) are recorded.
    4. Averaging Results: The performance metrics across all folds are averaged to provide a more reliable estimate of the model's generalization ability.

v. References:
    1. LightAutoML documentation
    2. LightGBM documentation
    3. CatBoost documentation

b. Describe how to reproduce the results with my source code files:

i. The first step was to execute “data_preprocessing.py”, then the training data would be preprocessed and exported two files named “train_X_process.csv” and “train_y_process.csv”.

ii. Second, the training model process is implemented in a Python script “train_classification_model.py”,

   1. Import Required Libraries and set parameters.
   2. Import train dataset
   3. Initializing the LightAutoML Task.
   4. Defining Feature Roles about target variable and columns to be dropped during training.
   5. Configuring and Initializing LightAutoML: Configures LightAutoML with a binary classification task.
   6. KFold Cross-Validation sets up 5-fold cross-validation to split the data into training and validation sets for each fold:
   7. Assigning Fold IDs:
      a. train_data['fold_id'] = -1: Initializes a new column fold_id with default value -1.
      b. kf.split(train_data): Splits the dataset into training and validation indices for each fold.
      c. train_data.loc[val_idx, 'fold_id'] = fold: Assigns the current fold ID to the validation samples.
      d. print(train_data['fold_id'].value_counts()): Verifies the distribution of samples across folds.
   8. Model Training and Out-of-Fold Predictions:
      a. automl.fit_predict(): Trains the model using 5-fold cross-validation and generates out-of-fold (OOF) predictions.
   9. Evaluating Model Performance on Each Fold and Calculating Average AUROC and macro F1-score:
      a. roc_auc_score: From sklearn.metrics import roc_auc_score and computes the AUROC for each fold.
      b. f1_score: From sklearn.metrics import f1_score and computes the macro F1-score for each fold.
   10. After training, the trained model would be saved for future use.

iii. Third, execute “test.py” to predict the result of the testing data using the trained model and you could observe the Top20 most important features. Then, export the predict result as “testing_result.csv” for Kaggle.

c. Problems I faced:

i. Initially, I used SMOTE to increase the amount of data whose label is 1 to the amount of data whose label is 0. However, I observed that the trained model by that kind of training dataset got an excellent performance on the validation dataset but a awful performance on the testing dataset. I guessed that the trained model overfit to the training dataset. Therefore, I decresed the amount of data whose label is 1 to only 20% of the amount of data whose label is 0. Fortunately, I got a better performance on the testing dataset.

3. Results & Analysis

a. Count Average AUROC and macro F1-Score with cross-validation method

i. Average AUROC:

Average AUROC=0.9550. The results of each AUROC of 5 folds are as follows:

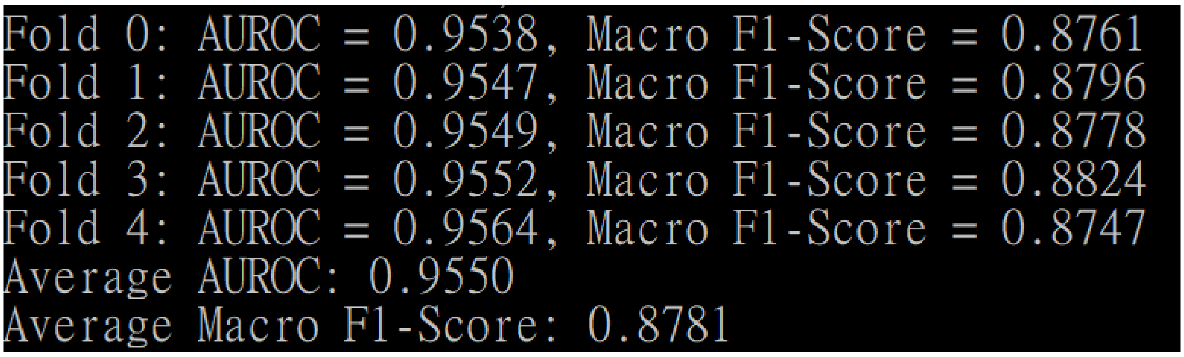
1. Fold 0: AUROC=0.9538
2. Fold 1: AUROC=0.9547
3. Fold 2: AUROC=0.9549
4. Fold 3: AUROC=0.9552
5. Fold 4: AUROC=0.9564

ii. Average macro F1-Score:

Average macro F1-Score=0.8781. The results of each macro F1-Score of 5 folds are as follows:

1. Fold 0: macro F1-Score=0.8761
2. Fold 1: macro F1-Score=0.8796
3. Fold 2: macro F1-Score=0.8778
4. Fold 3: macro F1-Score=0.8824
5. Fold 4: macro F1-Score=0.8747

iii. Screenshot my results (i.e., Average AUROC and macro F1-Score)

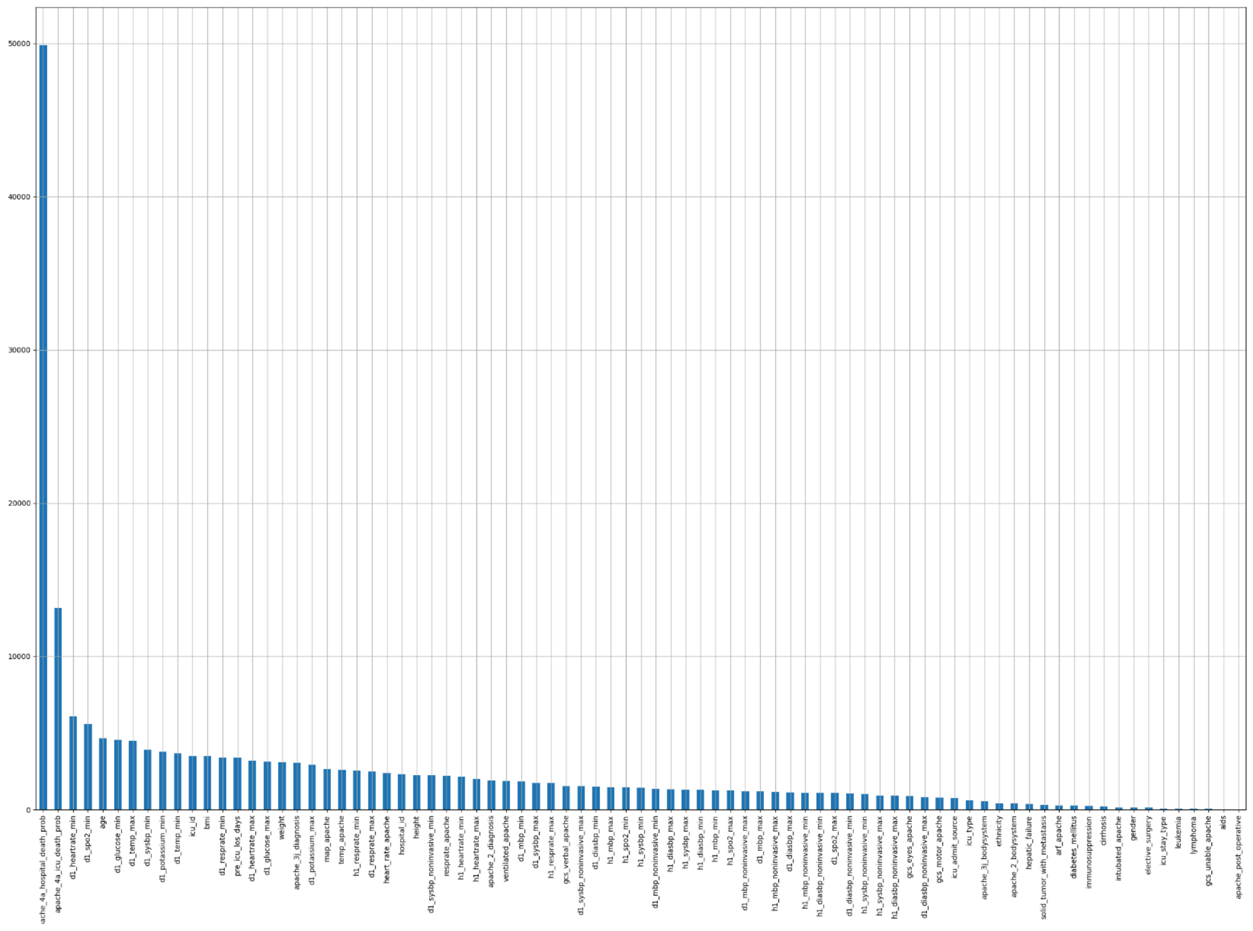


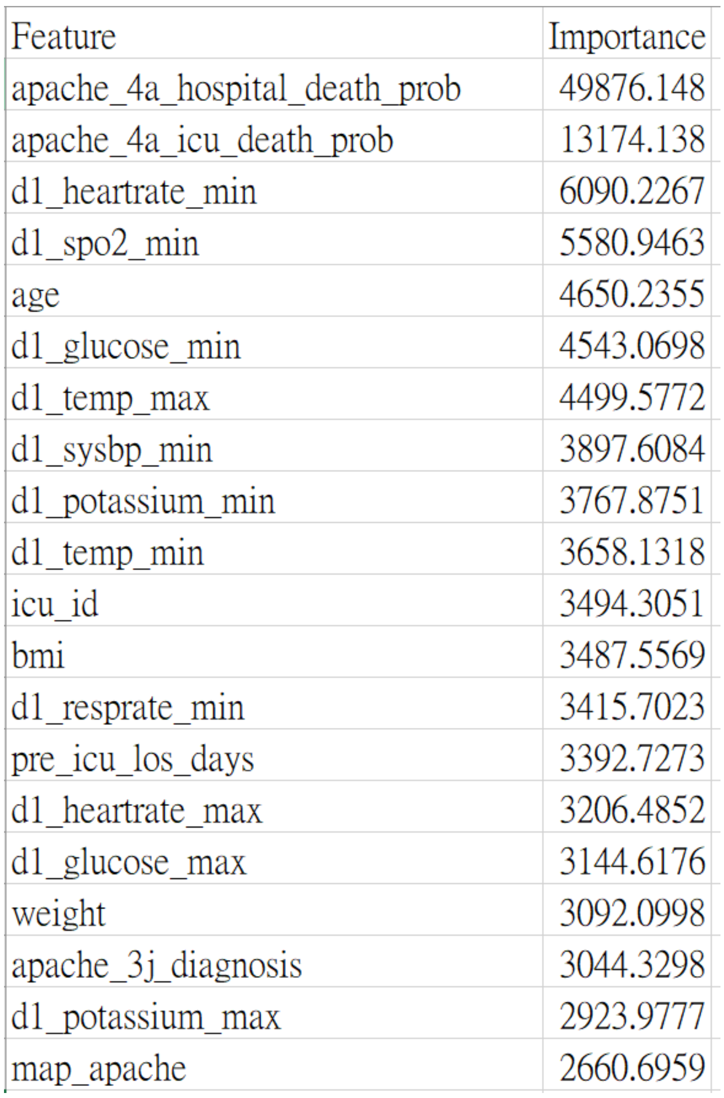
```
Fold 0: AUROC = 0.9538, Macro F1-Score = 0.8761
Fold 1: AUROC = 0.9547, Macro F1-Score = 0.8796
Fold 2: AUROC = 0.9549, Macro F1-Score = 0.8778
Fold 3: AUROC = 0.9552, Macro F1-Score = 0.8824
Fold 4: AUROC = 0.9564, Macro F1-Score = 0.8747
Average AUROC: 0.9550
Average Macro F1-Score: 0.8781
```

b. Explainable experiment

i. Show the Top 20 most important features:

1. Bar plot:





Feature	Importance
apache_4a_hospital_death_prob	49876.148
apache_4a_icu_death_prob	13174.138
d1_heartrate_min	6090.2267
d1_spo2_min	5580.9463
age	4650.2355
d1_glucose_min	4543.0698
d1_temp_max	4499.5772
d1_sysbp_min	3897.6084
d1_potassium_min	3767.8751
d1_temp_min	3658.1318
icu_id	3494.3051
bmi	3487.5569
d1_resprate_min	3415.7023
pre_icu_los_days	3392.7273
d1_heartrate_max	3206.4852
d1_glucose_max	3144.6176
weight	3092.0998
apache_3j_diagnosis	3044.3298
d1_potassium_max	2923.9777
map_apache	2660.6959

ii. Give some analysis and insights about the high-importance features
    1. apache_4a_hospital / icu_death_prob: The APACHE IVa probabilistic prediction of in-hospital / icu mortality for the patient which utilizes the APACHE III score and other covariates, including diagnosis. And the APACHE III score is a severity-of-disease classification system widely used in critical care settings to predict patient outcomes. It is calculated based on Acute Physiology Score, including 17 physiological variables (e.g., blood pressure, heart rate, temperature, arterial blood gases), Age Points, and Chronic Health Points. As a result, apache_4a_hospital_death_prob feature would lead to a significant impact on predicting if patients die.
    2. d1_heartrate_min: It reflects the patient’s minimum heart rate observed during the initial 24 hours. Patients with abnormally low values may have a higher risk of mortality. Therefore, the feature is important to predict if patients die.
    3. d1_spo2_min: It indicates the patient’s lowest oxygen saturation level measured using pulse oximetry in the first 24 hours. Because persistent hypoxemia within the first 24 hours often predicts the severity of the underlying condition and poorer survival rates. Therefore, d1_spo2_min is a strong predictor of mortality.
    4. age: It means the age of the patient on unit admission. Older patients tend to have more comorbidities (e.g., diabetes, hypertension) and reduced physiological reserve, making recovery slower and more complex. Therefore, age is a strong predictor of mortality.
    5. d1_glucose_min: The lowest glucose concentration of the patient in their serum or plasma during the first 24 hours of their unit stay70. Hypoglycemia can lead to neurological damage, seizures, and increased mortality. Hypoglycemia is associated with worse ICU outcomes, highlighting the importance of maintaining glucose homeostasis, so this is an important predictor of mortality.

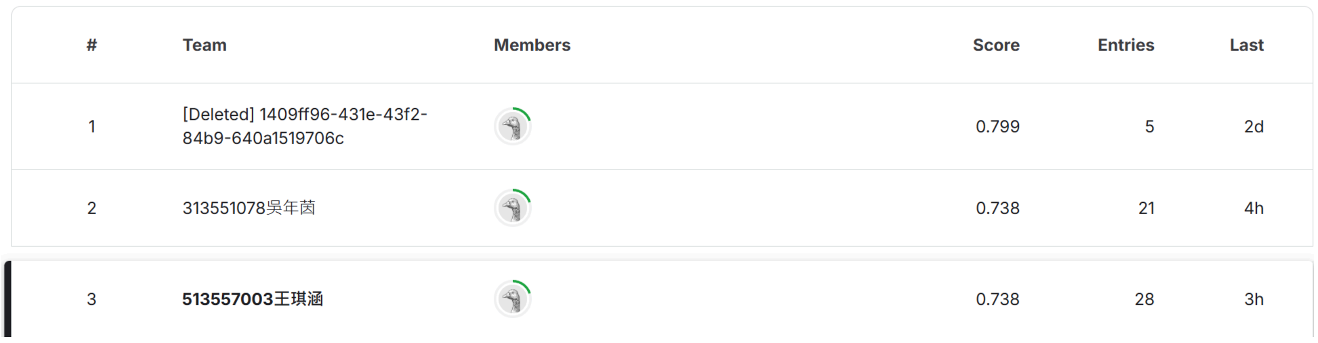
Part 2:

1. Classification performances on the Testing Set (in Kaggle competition)

a. My classification model is evaluated based on Macro F1-Score, and the score is 0.738

b. I got 3rd place in Kaggle competition.

c. Screenshot time at 2024/11/27 22:30



#	Team	Members	Score	Entries	Last
1	[Deleted] 1409ff96-431e-43f2-84b9-640a1519706c		0.799	5	2d
2	313551078吳年茵		0.738	21	4h
3	**513557003王琪涵**		0.738	28	3h