Data Mining Assignment 2
Classification

Bo-Han Chen (陳柏翰)
Student ID:312551074
bhchen312551074.cs12@nycu.edu.tw

Data Overview

The given dataset contains 44939 patients' information, including 81 features and 1 label representing whether the patient has died. Among the 81 features, 23 features are categorical and the rest are numerical.

Categorical Features

Categorical features contain two types of data, including *bool* and *object*. After visualizing the data, I found several features that are highly related to the result of death. The details are shown as follows:

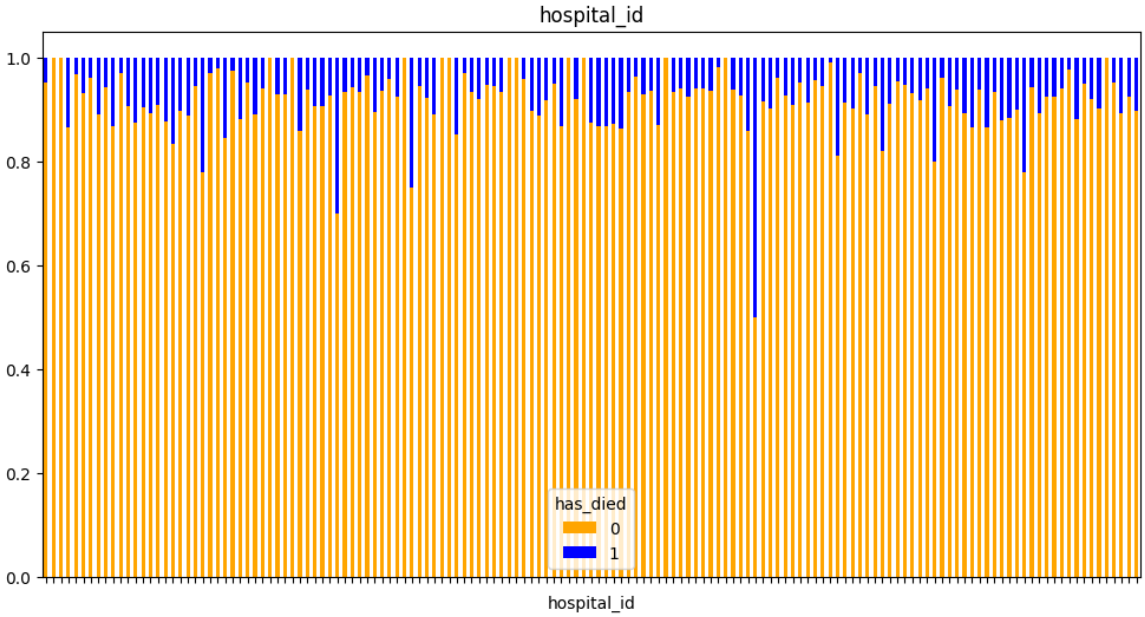

Figure 1: Percentage of Death in Different Hospital

From the figure 1 and figure 2, we can see that some of the hostipal and icu have higher death rate than others. So it's reasonable to assume that the hostipal and icu information are related to the result of death in this dataset, and it is necessary to keep these features.
Since there are some missing values in the categorical features, so we can first analyze the relationship between missing values and result of death, and then decide the way to transform the missing values. First, I analyze the percentage of missing values in each feature in figure 3. From the figure, we can see that the missing values accounts for proportion of 0.75% to 1.75% in most of the categorical features, now we can look into the features with high percentage of missing values. For feature *apache_2_bodysystem* and *apache_3j_bodysystem* and *ethnicity*, the relationship between missing values and death rate is shown in figure 4, figure 5 and figure 6 From the figure, we can see that the missing values in these three features still contains some information about the result of death, so it's not reasonable to simply drop these missing values. For the rest of categorical features that also contains missing values, the missing values are also accounts for a small proportion of death rate. Although these related death result represents a very small proportion of the whole dataset, it's still necessary to keep these missing values to avoid losing important information.

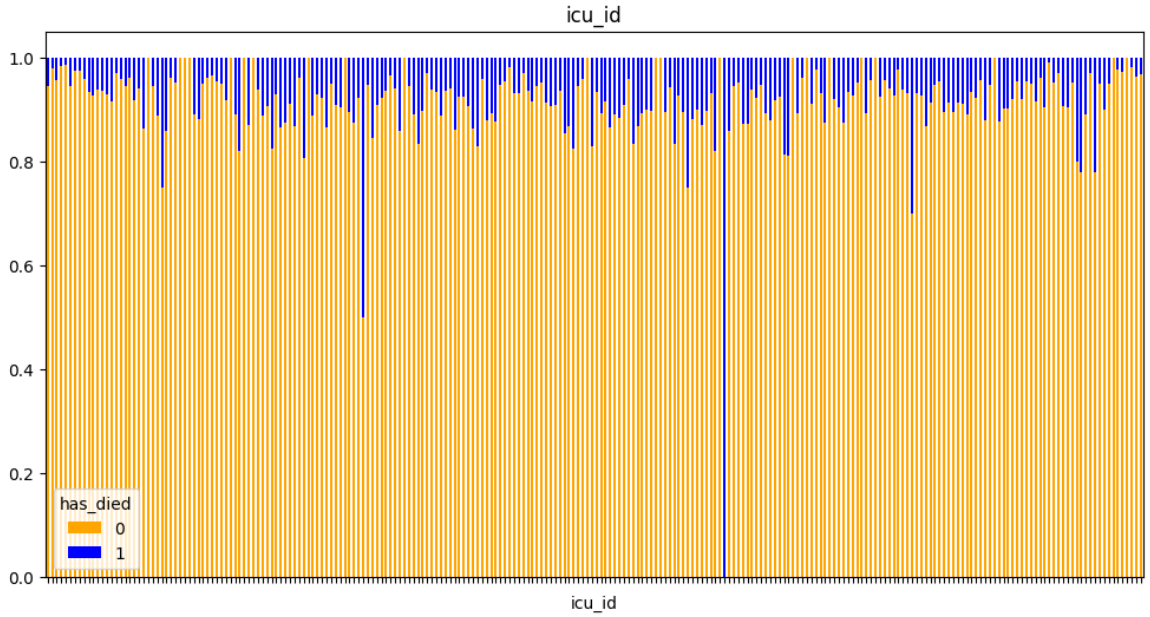

Figure 2: Percentage of Death in Different ICU

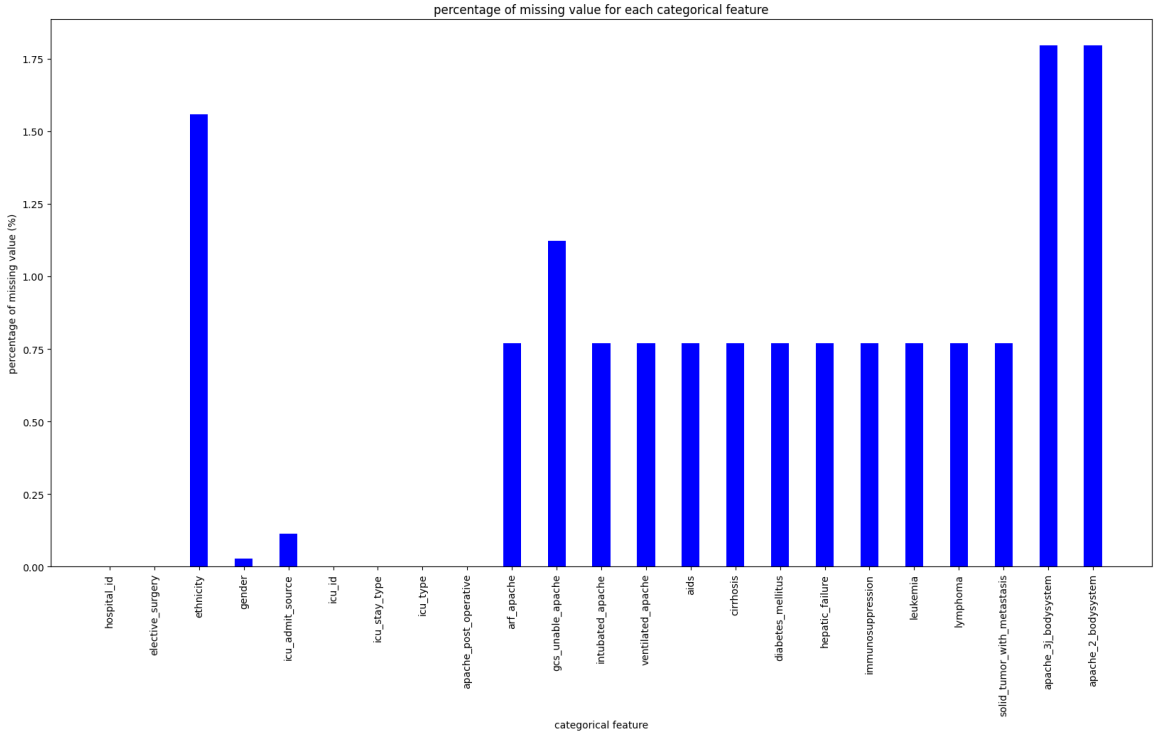

Figure 3: Percentage of Missing Values in Categorical Features

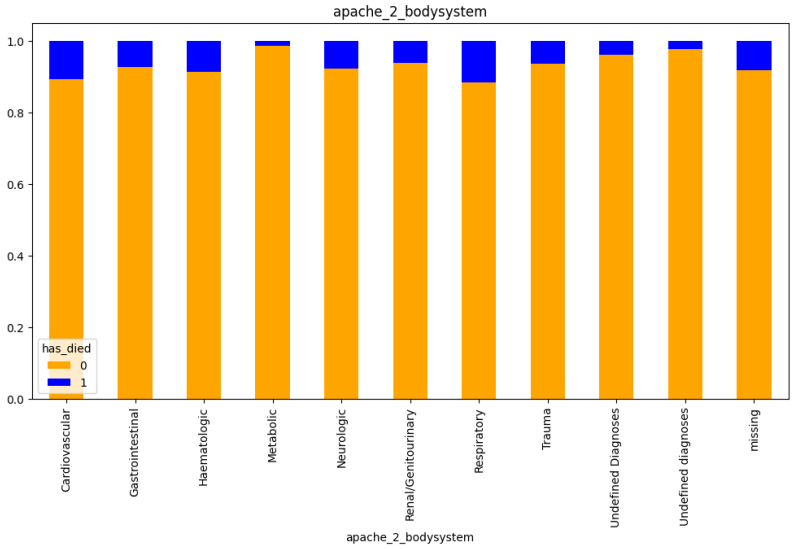

Figure 4: Percentage of Death in Different apache_2_bodysystem

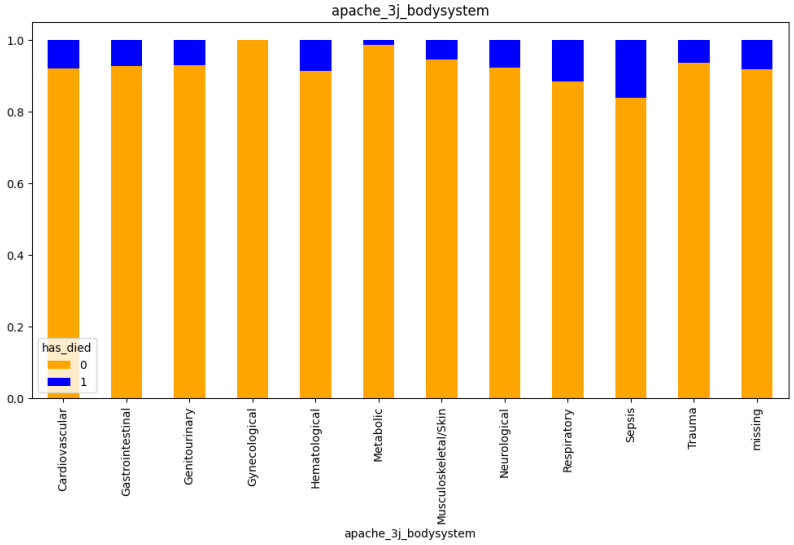

Figure 5: Percentage of Death in Different apache_3j_bodysystem

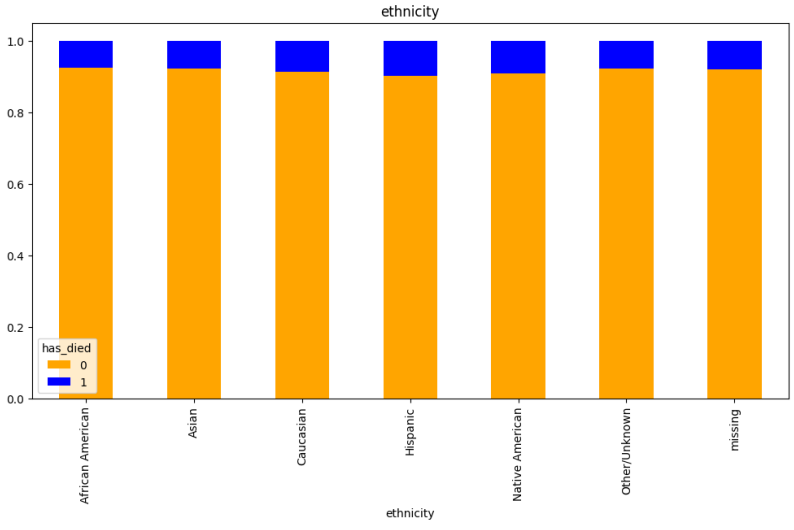

Figure 6: Percentage of Death in Different ethnicity

Numerical Features

For numerical features, I first analyze the percentage of missing values in figure 7. From the figure we can see the proportion of missing values in most of numerical features is much higher than the categorical features. For further analysis, I fill the missing values with the mean value of each feature, which can minimize the impact of missing values.

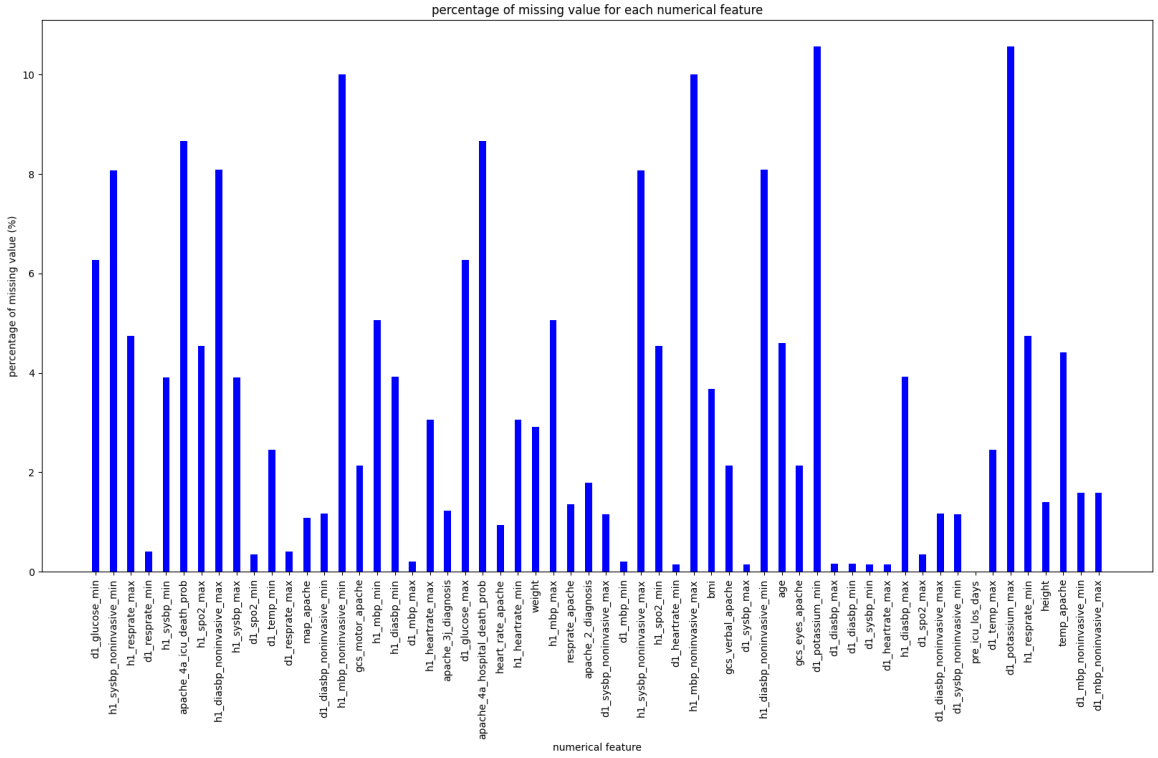

Figure 7: Percentage of Missing Values in Numerical Features

Figure 8 show the distribution of bmi in different result of death. From the figure, we can see the trend of bmi value among our dataset, and the percentage of death is quite similar in different bmi value. An other interesting discovery is that there are some abnormal distribution near the bmi value 70, which I think is the outlier of this dataset at the first glance. However, after further analysis, I found that the same distribution trend can be found in feature *weight* from figure 9, which means the abnormal distribution may be casued by realistic reasons, such as some diagnosis or treatment may cause the patient to gain weight.

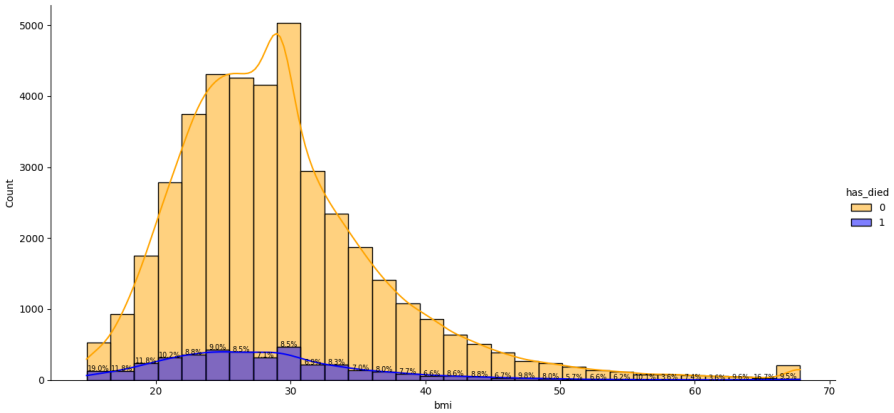

Figure 8: Distribution of bmi

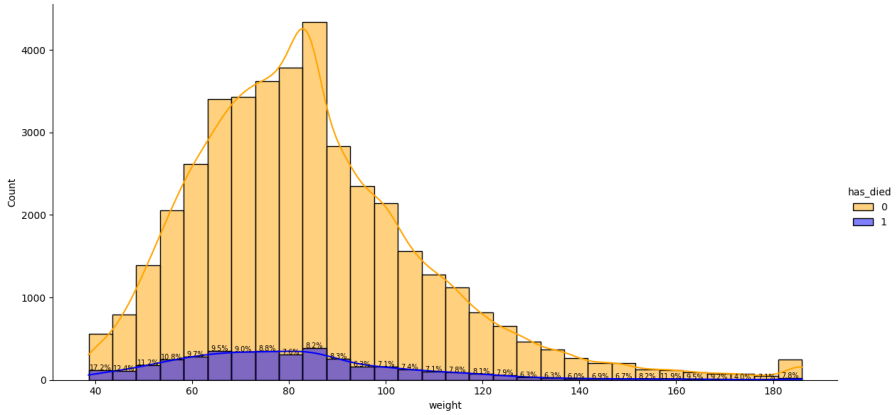

Figure 9: Distribution of weight

The discovery about outliers can be found in feature *apache_4a_hospital_death_prob* and *apache_4a_icu_death_prob*. From figure 10 and figure 11, we can see there are some negative values in these two features, which is obviously the outliers when representing the probability of death. To dealing with these outliers, I replace them with NaN values and use imputation for restoring, expect this operation can let the new value be more realistic. The details of imputation will be mentioned in the following section.

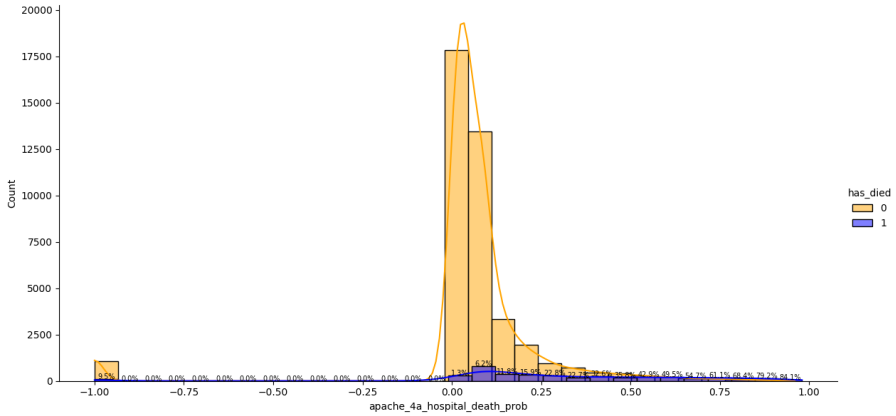

Figure 10: Distribution of apache_4a_hospital_death_prob

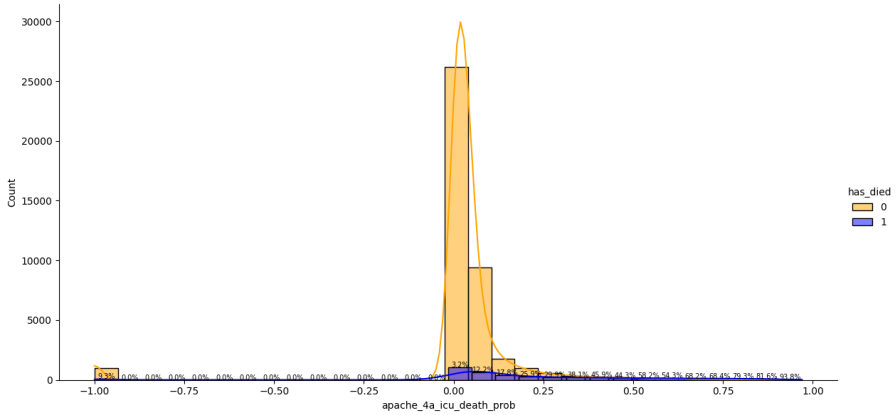

Figure 11: Distribution of apache_4a_icu_death_prob

Imbalanced Data

For this dataset, the proportion of death is only 8.63%, which means the dataset is highly imbalanced (see Figure 12). Since the imbalanced data will lead to frequency bias on the learning model, so

there are some solutions for training on a imbalanced data, which will be concluded in the following section.

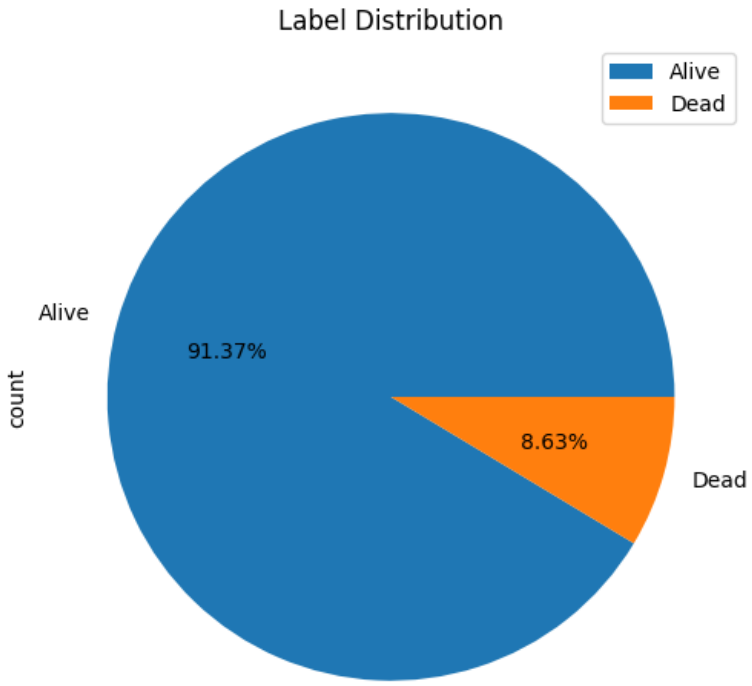

Figure 12: Distribution of Label

Data Preprocessing

Feature Encoding

For this dataset, I tried several encoding method for categorical feature transformation.

One-hot Encoding

One-hot encoding is the most common way to transform categorical features, but for the features with multiple categories, it will lead to sparse embedded data with high dimension, which ccause the curse of dimensionality.

Frequency Encoding

Frequency Encoding is an better way to avoid the curse of dimensionality comparing to one-hot encoding. The idea of frequency encoding is to replace the category value with the frequency of that category in the dataset. But for the features that each category with similar frequency, it will cause the problem of information loss.

Target Encoding

For target encoding, it will replace the category value with the mean value of the target value in that category. the advantage of target encoding is that it shows the relationship between each category and target value. But it may cause the problem of overfitting since the label information is directly used in the training process.

Imputation

For imputation, I tried several imputation methods to restore the missing values in the dataset. Since the missing values of categorical features accounts for a large proportion of the whole dataset, so it's important to find a proper way to restore the missing values.

Most-Frequent Imputation

Most-frequent imputation is a common way to restore the missing values of categorical features. Since for each categorical features, it's highly possible that the actual value is the most frequent value.

Mean / Median Imputation

For numerical features, mean / median imputation is a reasonable way to restore the missing values. Since the mean / median value can be close to the actual value of the missing values.

KNN Imputation

For the imputation techniques mentioned above, they are all completed according to the same single feature of the dataset. But it's more reasonable to restore the missing values according to the relationship between different instances, which considers other features when measuring the similarity between instances. KNN imputation is such a method like that, which first find the k nearest instances of the one with mmissing values, and then use mean value to restore the missing values. The advantage of KNN imputation is quite obvious since the basis of imputation is more accurate, but it also has the disadvantage of time complexity comparing to the other two methods.

Disscussion of Encoding and Imputation Method

For the imputation and encoding method mensioned above, I tried several combinations of them, and the result is shown in the table 1, which shows the internal K-Fold average Macro F1 score and AUROC score of each combination. The *Simple* imputation method means applying the median imputation for numerical features, and most-frequent imputation for categorical features. Experiments are all following the same training and testing process mensioned in experiment section, and the best result of each combination is shown in the table. We can see that KNN imputation method can reach higher performance than Simple imputation method, which means using the relationship between instances can help the model to learn better. For the encoding method, target encoding can reach higher performance than frequency encoding, which means the relationship between category and label is important for this dataset. According to the result, I choose the combination of *Target* encoding and *KNN* imputation for the following experiment.

Encoding	Imputation	Macro F1	AUROC
Frequency	Simple	0.728	0.734
Frequency	KNN	0.728	0.737
Target	Simple	0.730	0.737
Target	KNN	0.732	0.740

Table 1: Evaluation of Encoding and Imputation Method

Note: The result of this table are all evaluated on Persenal Computer (spec below), so the result may be slightly different from the result of the following section, which is evaluated on Kaggle Platform.

- OS: Windows 10 22H2
- CPU: Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
- python: 3.12.0

Feature Selection

Feature selection is an important step in data preprocessing, which can reduce the dimension of training data and help the model reach higher performance.

Correlation Coefficient

Furthermore, we can also use correlation coefficient to find the relationship between each feature and result of death, and then select the features that is highly related to the label. When selecting the features, we can also consider the correlation between features to avoid the problem of multicollinearity. For the correlation of each feature and label, I use biserial correlation coefficient to measure the relationship, which is suitable for the relationship between continuous and binary variables, figure 13 shows the result.

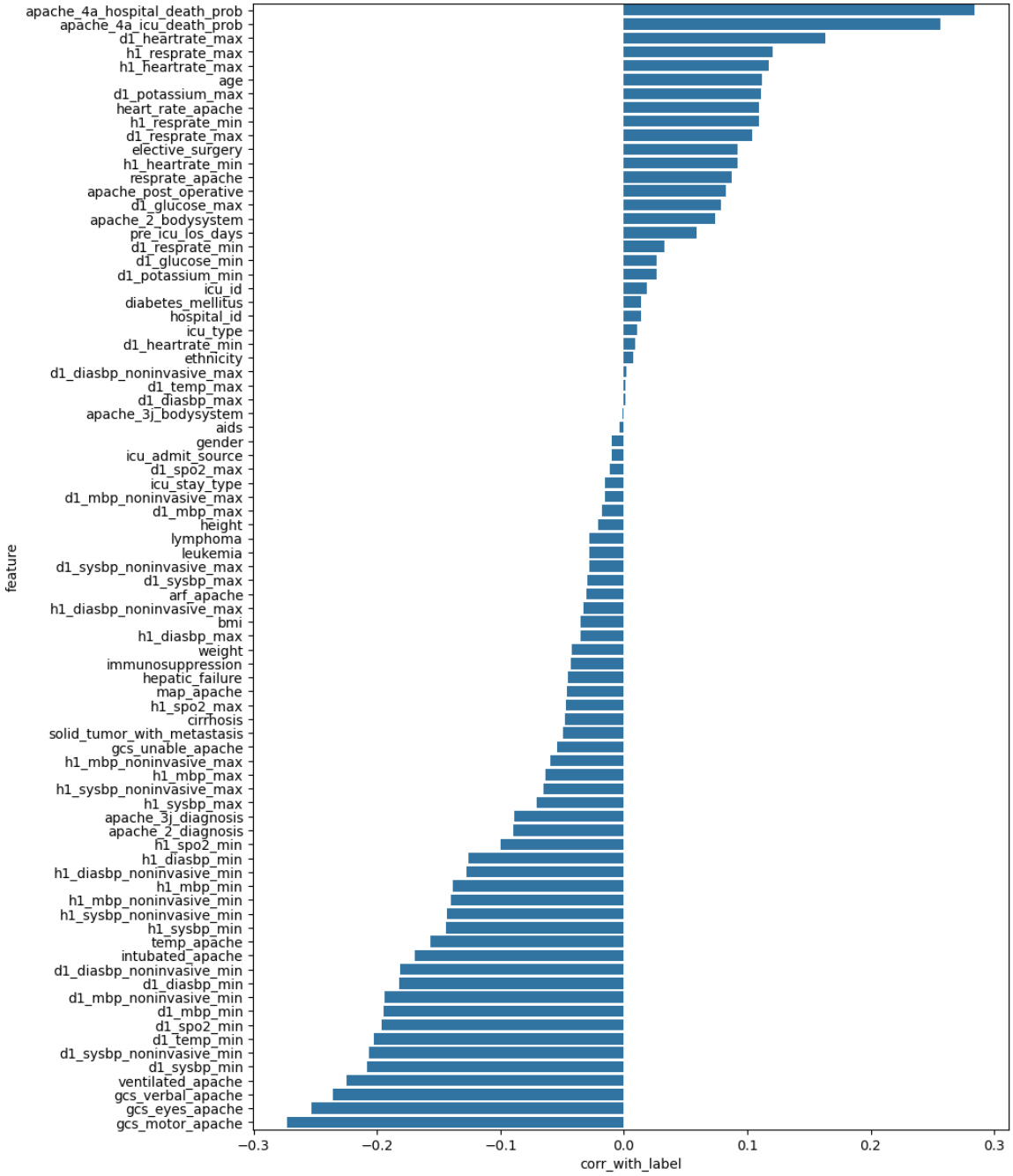

Figure 13: Biserial Correlation Coefficient

Pearson correlation coefficient is used to measure the correlation between continuous variables, so I use it for finding the correlation between features, figure 14 shows the result.

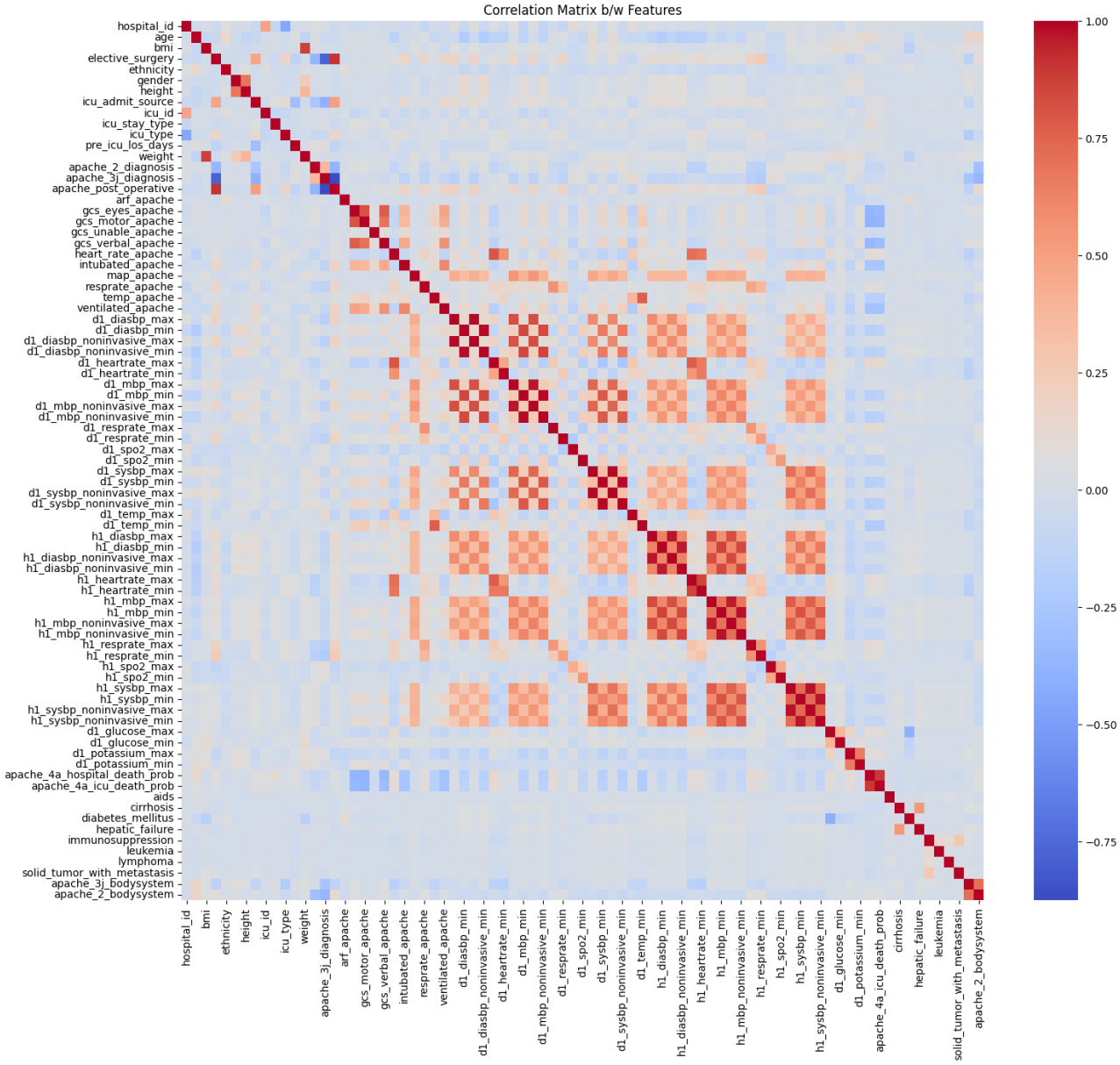

Figure 14: Pearson Correlation Coefficient

Sharpley Value

Sharpley value is a metric for measuring the importance of each feature, which is suitable for the classification problem. The idea of Sharpley value is to calculate the difference of the model performance before and after removing the feature. The feature with higher Sharpley value means it has higher impact on the model performance. Figure 15 and figure 16 shows the Sharpley value of top20 features before selection.

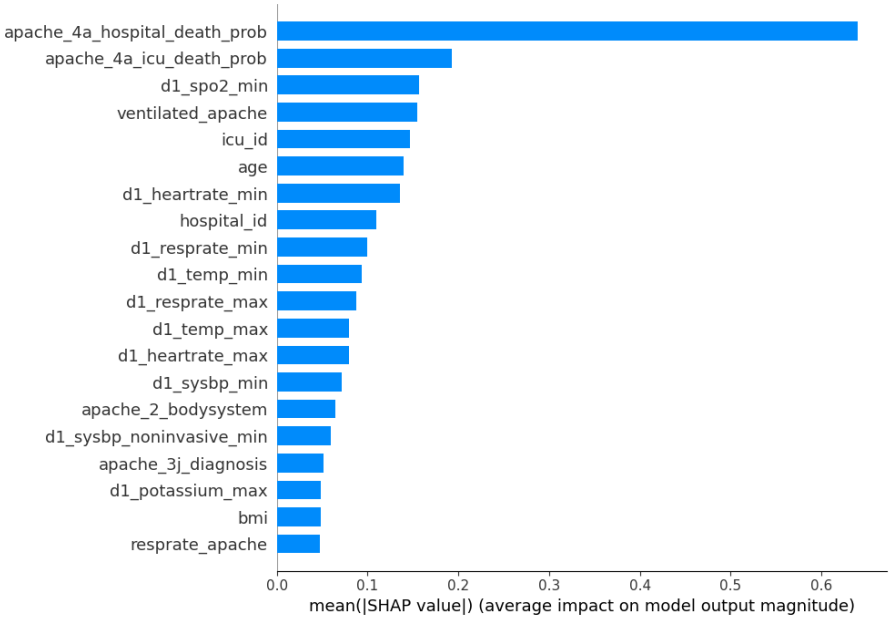

Figure 15: Top20 Mean Sharpley Value Before Selection

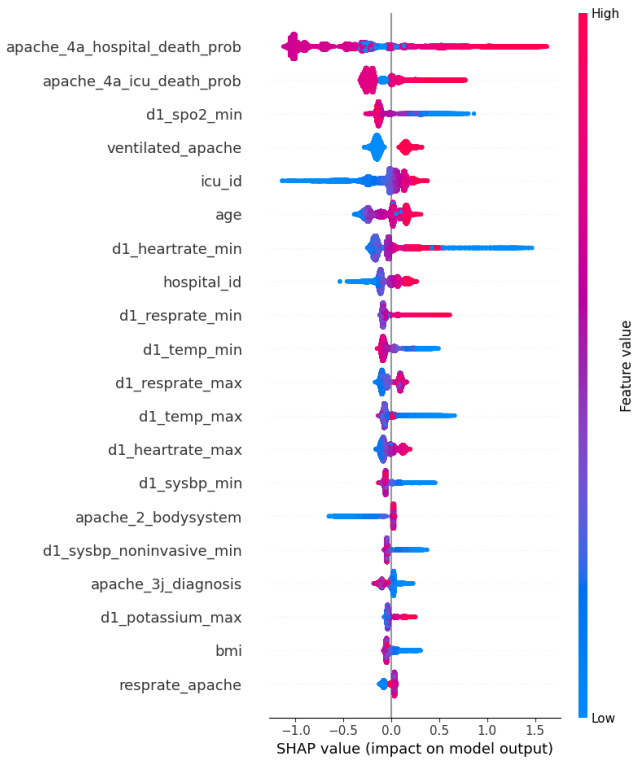

Figure 16: Top20 Sharpley Value Before Selection

In the figure, we can see the features with higher Sharpley value, such as *apache_4a_hospital_death_prob*, *apache_4a_icu_death_prob*, *d1_spo2_min*, etc. These features are also have high correlation with the result of death, but we cannot simply choose these features since they might cause multicollinearity problem with other features that also have high Sharpley value. So I consider Sharpley value, biserial correlation to find the features that are highly related to the result of death and contribute greatly to the model performance, And remove the features that are highly correlated with other features using Pearson correlation coefficient. Additionally, the dataset is highly imbalanced, so I use undersampling to balance the dataset for selecting features. Since the sampling method are totally random, so I repeat the sampling process for several times, and choose the features that are selected for most of the time. Finally I choose 22 important features for training the model. Figure 17 and figure 18 shows the Sharpley value of each feature after feature selection.

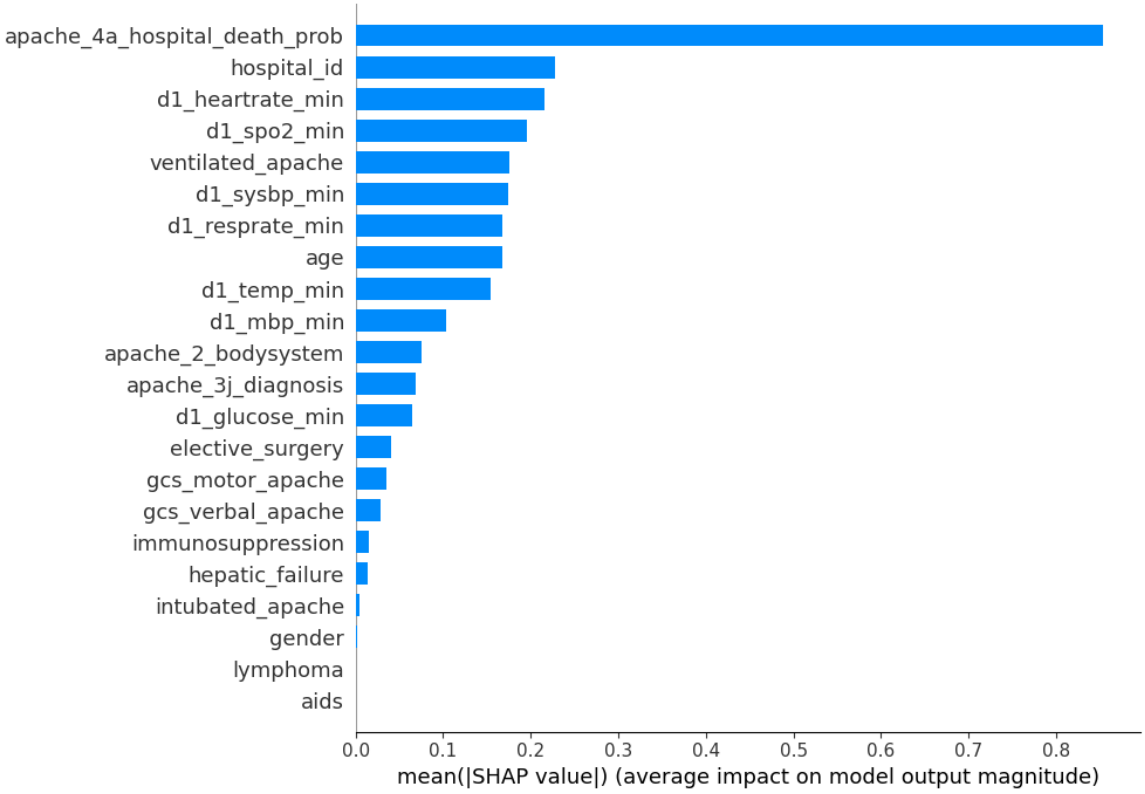

Figure 17: Mean Sharpley Value After Selection

Now we can see the features with higher Sharpley value and biserial correlation coefficient are selected. Some features with high correlation with other features are removed, such as *apache_4a_icu_death_prob*, although it has high Sharpley value, but it has high correlation with *apache_4a_hospital_death_prob* (according to figure 14), so it's removed. Other features are included in the feature selection for obvious reasons, such as *hospital_id*, which we can see from figure 1 that the death rate differ dramatically in different hospital. There are some features that are selected but they don't have high correlation with the result of death or Sharpley value, such as *gender*, *lymphoma* and *aids*. So I think I can try further feature selection to find the best combination of features. In order to do that, I use the *SequentialFeatureSelector* function from *mlxtend* package, which can find the best combination of features according to the given evaluation metrics (Marco F1 score).

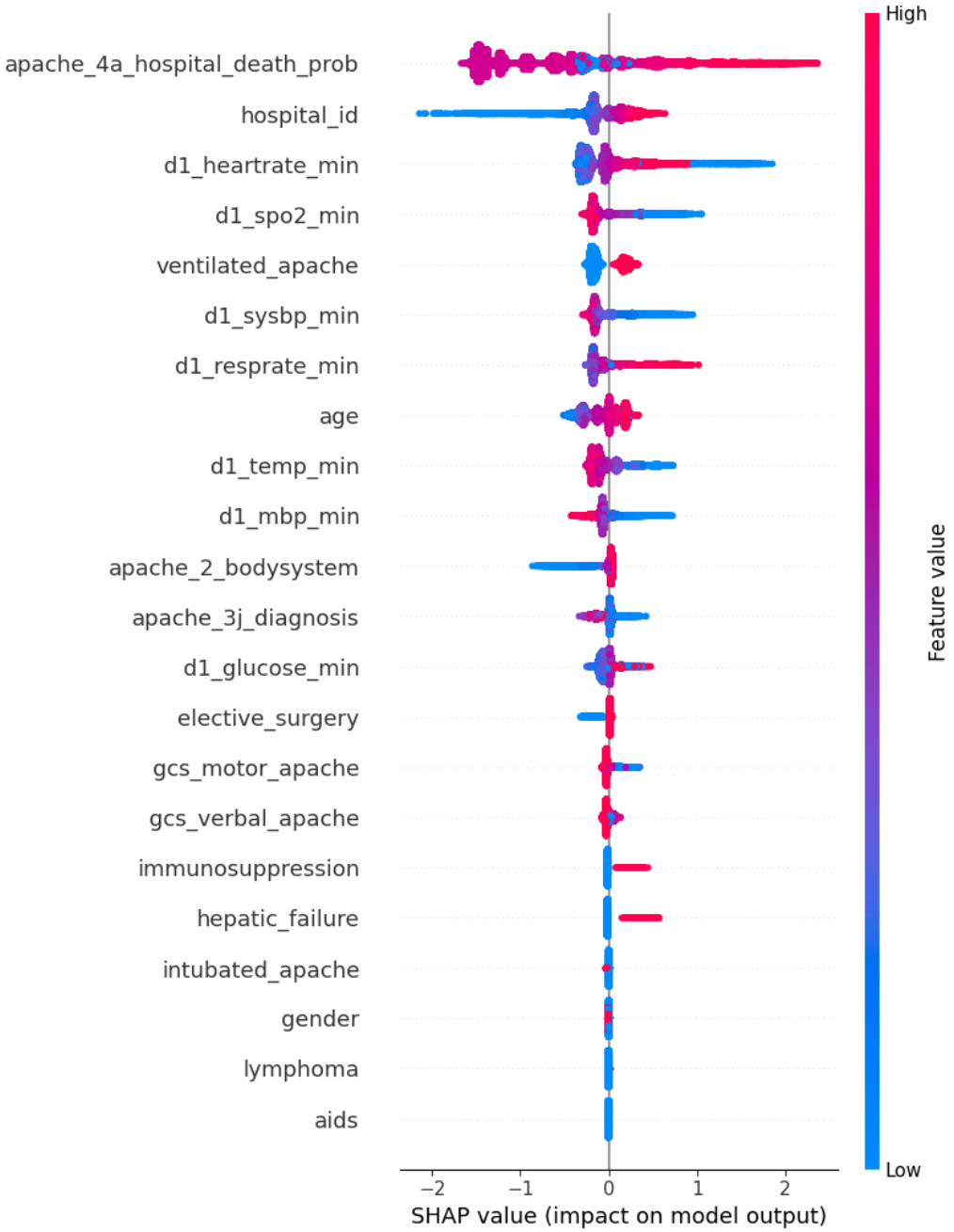

Figure 18: Sharpley Value After Selection

Classification Model

Model Selection

The classification model I used for this assignment is XGBoost [1], which combines the advantages of both bagging and boosting. For bagging, it uses the idea of random forest to train multiple weak learners (trees) with different subset of training data, and each new tree will correct the error of the previous tree, which is the idea of boosting. The advantage of XGBoost is that it can avoid the overfitting problem of boosting with the regularization term and pruning technique, which can also reduce the time complexity.

Hyperparameter Tuning

For hyperparameter tuning, I use the *GridSearchCV* function from *sklearn* package. With this function, I can find the hyperparameter profile that can reach the highest performance within the internal K-fold cross validation.

Training with Imbalanced Data

For imbalanced data, I tried oversampling method such as SMOTE, but it did not improve the performance of the model, worsely, it will cost more time for training since the dataset is almost 2 times larger. So I choose to adjust class weight to deal with this problem. The idea of adjusting class weight is to increase the weight of the minority class, which can let the model pay more attention to the minority class. I tried several weight ratio, and find that the ratio 3:1 can reach the best performance.

Model Evaluation

For model evaluation, I follow the instruction of the assignment, which uses the Macro F1 score and AUROC score as the evaluation metrics. These two metrics are both suitable for imbalanced data, which can truely reflect the performance of the model.

Experiment

Experiment Environment

- Platform: Kaggle Notebook
- Python 3.10.2
  - pandas 2.0.3
  - sklearn 1.2.2
  - mlxtend 0.23.0
  - xgboost 2.0.2

Experiment Setting

- k = 5 for KNN imputation
- k = 5 for internal K-Fold cross validation (split by *sklearn.model_selection.KFold*)
- XGBoost (hyperparameter found by GridSearchCV)
  - colsample: 0.8
  - gamma: 0.1

  - learning rate: 0.1
  - max_depth: 3
  - min_child_weight: 1
  - n_estimators: 12000
  - random_state: 42
  - scale_pos_weight: 3
- *SequentialFeatureSelector* from *mlxtend* package (hyperparameter found by GridSearchCV)
  - k_features: 15

Experiment Result

- Internal K-Fold (Running on Kaggle Notebook)
  - Macro F1 score: 0.723
  - AUROC score: 0.730
- Test data (Evaluated by Kaggle Competition)
  - Marco F1 score: 0.735

Note: The score of this section is all evaluated on Kaggle Platform, so the result may be slightly different from the result of table 1.

How to Reproduce the Result

- Import the ipynb file to Kaggle Notebook
- Modified the path of dataset in the fourth cell (file name) last cell (path)
- Run the code in order
- The result will be shown in the last cell

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.