**Predicting AKI Stage Using Machine Learning Models on MIMIC-IV Data**

The dataset contains first-day ICU patient data from MIMIC-IV. Features include ID (unique patient identifier), hospitality mortality (a binary indicator of whether the patient died during hospitalization), gender, admission age, etc. This dataset originally included 165 features of 50920 patients. This assignment aims to build models to predict patients’ AKI stages by different features. This report will describe how I selected features, built up, and optimized models in detail.

1. **Feature Selection Strategy**

High dimensionality can allow models to learn more in training. However, it can also cause a curse of dimensionality, meaning that as feature dimension space increases, the distance between data points becomes more and more similar, which makes algorithms hard to distinguish different samples. Over-dimensionality also brings problems like overfitting and expensive computational resources. Thus, it is important to conduct feature engineering before a model is constructed. I combined null value selection, variance selection, correlation selection, and logistic regression selection in feature engineering. Noticeably, all operations were conducted on the training set and transferred to the test set to avoid data leakage. Null values in the training set were replaced by training medians on groups, while null values in the test set were replaced by the whole test medians.

* 1. ***Null Value Selection***

There are 160 features with null values, except id, hospital mortality, AKI stage, gender, admission age, and race. Since features are randomly missing, keeping them with too many null values is unnecessary. For feature deletion rules, I set the threshold to be 0.5, meaning that if the proportion of null value in a feature exceeds 50%, the feature should be deleted. This threshold choice is because when a feature contains more than half of the samples with null values, this feature may become useless because we have to replace these null values with other statistics, which causes too much bias.

* 1. ***Small Variance Selection***

Variance is an indicator of whether a feature is representative or not. If the variance is high, it means a feature contains much information; otherwise, it means each sample under the feature is similar and the feature cannot distinguish and contribute to the target in machine learning, to a large degree. Random tests showed that a threshold from 0.1 to 0.5 gains the best AUC in logistic selection. When the threshold exceeded 1, features and AUC decreased dramatically, from 0.75 to 0.67. In this part, I set the threshold to be 0.5. Any feature with a variance smaller than this threshold was removed. After the null value and small variance selections, 75 features were kept.

* 1. ***Correlation Selection***

Many features in this dataset are closely related. Many indicators have their minimum, maximum, and mean as features. For example, the heart rate has its minimum, maximum, and mean. However, these features could be reduced since they share one mathematical expression. By definition of Pearson's correlation, a strong correlation indicates correlation coefficients larger than 0.6. Therefore, with the help of a heat plot, I deleted features with a correlation coefficient of more than 0.6. After this step, 41 features were kept.

* 1. ***Logistic Regression-Based Greedy Selection***

Although the feature dimension has been greatly reduced after 3 steps, there may not be enough evidence to tell that all the remaining features contribute to the AKI stage, which is the target of the prediction task. To solve this challenge, I constructed a logistic regression model to filter important features. First, multiple AKI stages (0 ~ 3) were transferred to the binary stage (0, 1), where 0 meant the patient did not experience AKI and 1 meant the patient went through AKI (stage 1, 2, 3). Features should be standardized and sent to the logistic regression model. K-cross validation was applied when adding features one by one for training (k = 5) to avoid data leakage. The whole training set was split into 5 folds: 4 folds for training and 1 for validation. When one feature was added to the empty original feature set, the AUC score would be calculated on the 1-fold validation set. This process lasted 5 times and each time there was a different validation set split from the whole training set. Ultimately, the average AUC score would be calculated over the 5-fold cross-validation. As each feature was added in, if the average AUC score increased lower than a threshold, then feature selection would be stopped. The threshold was set to 0.001, neither too small nor too large so that a suitable number of features could be included. Results showed that the best AUC score was 0.75 and 18 features were kept. They were: maximum partial pressure of carbon dioxide, Glasgow Coma Scale motor, minimum blood urea nitrogen, admission weight, minimum systolic blood pressure, admission age, maximum body temperature, minimum partial pressure of carbon dioxide, maximum systolic blood pressure, maximum partial thromboplastin time, minimum absolute neutrophil count, maximum potassium from lab tests, maximum oxygen saturation in the blood, maximum heart rate, maximum international normalized ratio, minimum chloride from lab tests, maximum partial pressure of oxygen, minimum partial pressure of oxygen.

1. **Model Implementations**

In this part, I implemented three tree-based models: random forest, XGBoost, and lightGBM. Although they were all generated from the decision tree, details were different. I also optimized the three models and saw how they performed on this prediction task.

* 1. ***Random Forest***

Random forest is an algorithm that solves binary or multi-class classification problems. Random Forest operates using the bagging (Bootstrap Aggregating) technique. The algorithm randomly selects multiple subsets (with replacement) from the training dataset and Each subset is used to train an independent Decision Tree. Each decision tree only considers a random subset of features when splitting nodes, which ensures that the trees are diverse and reduces correlation among them. Then, a large number of decision trees are trained in parallel, each making independent predictions. The final prediction is determined by majority voting (i.e., the class that most trees predict is selected) in classification.

To optimize the random forest model, I used a randomized grid search within the following parameters: n estimators (100, 200, 300), maximum depth (10, 20, 30, None), minimum sample splits (2, 5, 10), minimum sample leaves (1, 2, 4), and maximum features function (sqrt, log2). Under 5 cross-validation folds, optimal parameters were: [n estimators = 300], [minimum sample splits = 10], [minimum sample leaves = 1], [maximum feature function = ‘sqrt’], and [maximum depth = None]. The class weight was also set to balance. The optimal accuracy on the test set was 0.36.

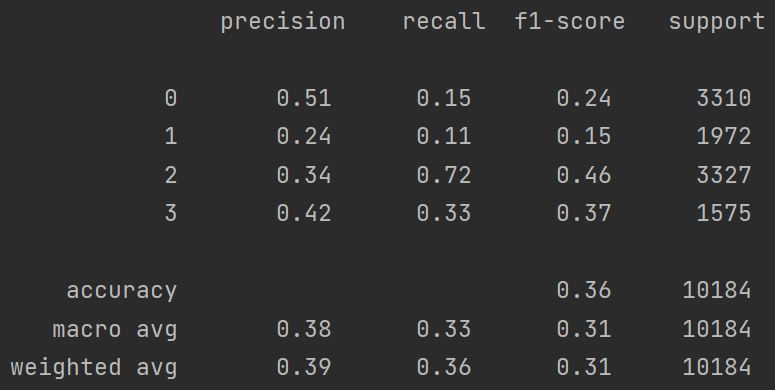


Figure 1: Random Forest Classification Report

From the classification report, it was obvious that among the four AKI stages, stage 0 had the best overall performance. For stage 1, the precision and recall were low, meaning that the model missed many actual stage 1 instances. This might indicate a class imbalance or difficulty distinguishing class 1 from others. The stage 2 had the highest recall.

* 1. ***XGBoost***

The XGBoost is an implementation of gradient boosting that improves training speed, performance, and scalability. The model starts with a simple prediction (e.g., the mean of the target variable). Each new tree corrects the errors of the previous trees by learning from the residuals (differences between actual and predicted values) and the trees are added one at a time, and each tree is weighted to minimize the loss function. After that, it comes to gradient descent optimization. Instead of fitting trees to the raw target values, XGBoost fits them to the negative gradient of the loss function. A learning rate (η) is applied to each tree’s contribution, preventing the model from overfitting, while L1 & L2 regularization help control complexity and prevent overfitting and tree pruning removes unnecessary splits to keep the model simple and efficient. Lastly, the final prediction is made by summing the predictions from all trees.

Using the same parameters random grid search on 5 cross-validation, I found the best parameters were: [n estimators = 200], [minimum sample split = 5], [minimum sample leaves = 4], [maximum feature function = ‘sqrt’], and [maximum depth = None]. The optimal accuracy on the test set was 0.38.

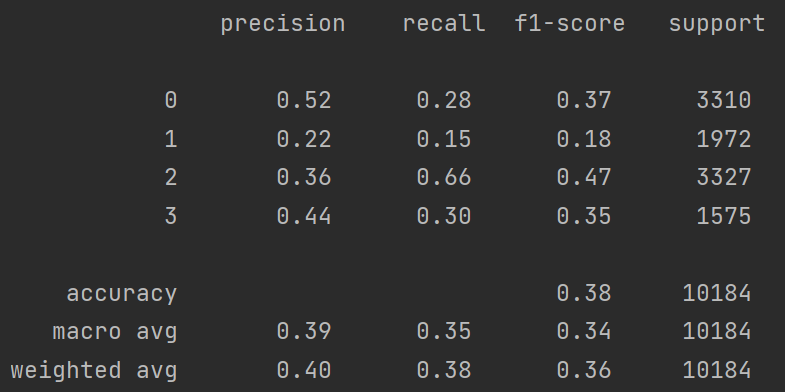


Figure 2: XGBoost Classification Report

The XGBoost classification report illustrated that precision and recall in each AKI stage were more balanced than that of random forest. Similarly, stage 0 had the best overall performance and stage 1 performed the worst. Recall for stage 2 was the highest among the other 3 classes.

* 1. ***LightGBM***

LightGBM is a gradient boosting model designed for efficiency and speed while maintaining high accuracy due to its histogram-based learning and optimized data structure. It also needs less memory than traditional tree-based machine learning models when facing large datasets. XGBoost applied many innovative ideas. Instead of evaluating every possible split across all feature values, LightGBM bins continuous features into histograms and finds the best split based on those bins. LightGBM grows trees leaf-wise, meaning it splits the leaf with the largest loss reduction, which was completely different from traditional GBDT’s level-wise grow mode. Besides, in standard GBDT, all samples are used to compute gradients, which is computationally expensive. However, GOSS selectively keeps samples with large gradients (harder-to-fit samples) and random samples from small-gradient samples (easier-to-fit samples) in XGBoost.

The random grid search on 5 cross-validation showed that the best parameters were: [number of leaves = 50], [n estimators = 300], [minimum child samples = 10], [maximum depth = -1], [feature fraction = 0.8]. The optimal accuracy on the test set was 0.35.

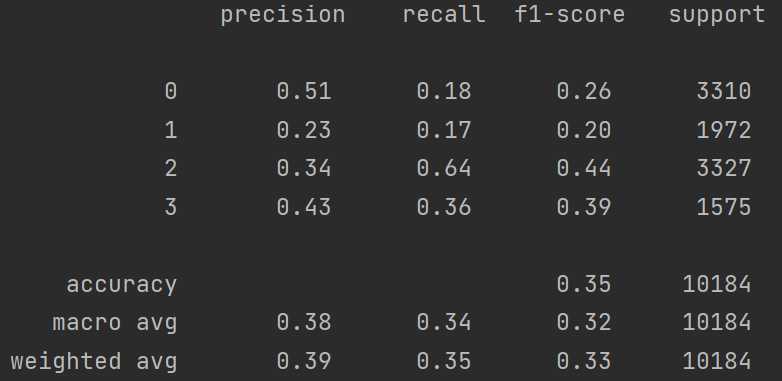


Figure 3: LightGBM Classification Report

The lightGBM classification report showed that similar to the previous two models, the stage 0 had the best performance, while the stage 1 performed the worst with the lowest precision.

1. **Summary**

This report proposed that by using null value selection, small variance selection, correlation selection, and logistic regression-based greedy selection, features could be extracted to the greatest degree. To get rid of data leakage, the whole dataset must be split into a training set and a test set from the very beginning and all the feature engineering work should be done on the training set then transferred to the test set by pipelines. All in all, among the three models, the XGBoost model performed the best. All results showed that stage 0 was the most precisely predicted one compared to the remaining three stages and the stage 1 performed the worst.

For coding, please visit <https://github.com/JimmyZzj/SPH-Assignment>