STATS 503 Data Challenge Summary

Xiaoyang Sheng, 01680797

1 Introduction

In this data challenge, data were collected during the stay of a patient in ICU to predict whether the patient will develop sepsis (0 for no sepsis and 1 for sepsis). The data consist of records from 21634 patients and has been split into a training set (with 15144 patients) and a test set (with 6490 patients). Outcomes are provided for the training set, and are withheld for the test set.

2 Data Preprocessing

2.1 NaN filling and Feature selection

Since there are many null values in the records, they should be filled to keep the model performance. For those records are missing between certain time stamps, I use the forward filling stragety, which means the value will be filled as the former record value if the former one is not missing, which is reasonable under the health detection in ICU. However, for those missing values not having a non-null previous records or not having records at all, they will not be filled at all.

2.2 Summary of records

Since every patient has multiple records for hours, and the number of records of each patient varies a lot, from several hours to hundreds of hours, which is difficult to regularize them into the same size. Therefore, to use these data, I use the summary of the data of each patient, which are the mean of all the variables and the difference of the maximum and the minimum value of all the variables, named as 'xxx_mean' and 'xxx_diff'. This is because by In this way, there is only one record for each patient, and the data is greatly reduced, which is good for model simplicity and the training cost. Meanwhile, the variation and basic characteristics of the records are still preserved.

2.3 Feature drop

After the procedures above, I get the data with each patients. After some simple EDA and summary, I decided to drop columns: 'Unit1', 'Unit2', 'Bilirubin_direct_mean','Bilirubin_direct_diff', 'EtCO2_mean', 'EtCO2_diff' , 'TroponinI_mean', 'TroponinI_diff', 'Fibrinogen_mean', 'Fibrinogen_diff', for they only have less than 3500 non-null values in the total of 21634 samples, even after the forward NaN filling. For other variables that have little missing values, I keep them unchanged, since I use the Xgboost classifier, which is a model with good performance under missing values.

3 Model Training

In this data challenge, I use Xgboost classifier, which is a gradient boosting decending tree-based classification model. It turns to perform well under many scenes, and it supports the task under some missing values.

3.1 Train-validation split

Apart from the test set, I split the rest data into train sets and validation sets, by dividing both the positive samples and negative samples with 30% as validation.

3.2 Cross Validation and Initial Model Setup

To find the best parameters of the model, I use 5-fold cross validation. The candidate of the params are shown as following:

max_depth	3,5,7,9
learning_rate	0.01, 0.02, 0.04, 0.06, 0.08, 0.1
colsample_bytree	0.5, 0.6, 0.7, 0.8, 0.9

Together with n_estimators=220 and the scale_pos_weight = the number of negative(0) samples/the number of positive(1) samples. Since the data set is greatly imbalanced with heavy negative samples, and the distribution of the train and set samples are similar, I add the scale_pos_weight param to make the model better performed under the unbalanced data samples.

3.3 The CV result and validation performance

After cross validation, the best params is set as 'colsample_bytree': 0.5, 'learning_rate': 0.06, 'max_depth': 9. Together with other default settings, I apply the model on the validation set. The **validation set** performance indicators are shown below:

AUC	0.8850818481948758
BER	0.26901537966106
precision_score	0.7372262773722628
model_score	0.9066490532804932

4 Program Setup

4.1 file structure

-final_data
——final_data_md.csv
——test_nolabel.csv
——test_set.csv
——train_set.csv
——train_outcome.csv
——x_all
————1.txt
————2.txt
————...
-main.py
-data_clean_merge.py
-model_train.py
-model_apply.py
-log.txt
-xgb_model.json
-test_result.csv

4.2 Quick Start and File saving

To run all the procedures, run main.py. Otherwise, you may run any one of the rest three py files to run the corresponding tasks.

The validation performance indicators are saved in the log.txt, the model is saved in json file. The final result of the prediction is saved in the **test_result.csv**.

All the files and details could refer to:

`https://github.com/xiaoyang-sheng/Prediction-of-Sepsis-from-Clinical-Data`