Mortality of VV-ECMO for Obese Patients

Sean Han Byul Lee

2025-07-19

Data Preprocessing

The excluded variables below (e.g., wbc_avg, etc.) either missing too many data points or redundant to other variables.

```
df <- read.csv("~/Downloads/rediscover_ecmo_vv_v3.csv")

df <- df[!is.na(df$bmi_avg), ]
df <- df[, !(names(df) %in% c("wbc_avg", "lactate_avg",
                               "fibrin_avg", "sofa_neurological_score",
                               "patient_site", "site_description", "gender",
                               "race", "ethnicity", "visit_start_date",
                               "visit_ecmo_start_date", "visit_ecmo_end_date",
                               "death_date"))]
```

Missing Imputation: Random Forest Imputation

Random forest was selected as the imputation method for this VV-ECMO dataset due to its ability to handle complex, nonlinear relationships and mixed data types—common characteristics of clinical data involving vital signs, lab values, treatments, and demographic factors. Unlike traditional imputation methods that rely on linear assumptions, such as Multivariate Imputation by Chained Equations (MICE), random forest can capture intricate interactions and variable importance hierarchies without assuming a specific parametric form. MICE, while flexible and commonly used in epidemiological research, may struggle when variables have nonlinear effects or when high-order interactions are important, and it can be sensitive to model specification and convergence issues in high-dimensional settings. In contrast, random forest imputation is robust to outliers, less sensitive to model mis-specification, and can yield more accurate estimates in datasets like ours with many binary indicators (e.g., medication administration) and skewed clinical variables (e.g., ferritin or bilirubin levels). Its non-parametric nature and empirical accuracy make it a strong choice for generating realistic and consistent imputations in this critical care dataset.

```
set.seed(123)
df <- missForest(df)$ximp
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?

## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry = mtry,
## : The response has five or fewer unique values.  Are you sure you want to do
## regression?
```

Cleaning Up Variables & Define Predictors and Outcome

```r
# Obesity indicator (BMI >= 30)
df$obese <- ifelse(df$bmi_avg >= 30, 1, 0)

# Convert numeric categorical variables to factors
df <- df %>% mutate(across(c(visit_occurrence_id, gender_concept_id, race_concept_id, ethnicity_concept

# Remove redundant variables if necessary (e.g., IDs)
df <- df %>% select(-visit_occurrence_id, -died_within_30_days_of_ecmo)

# Define predictor matrix and outcome
X <- model.matrix(died ~ . -1, data = df)
y <- df$died
```

Feature Selection with LASSO

LASSO is well-suited for feature selection in our VV-ECMO mortality prediction study because it effectively handles high-dimensional clinical data where many variables may be correlated or only weakly associated with the outcome. By applying an L1 regularization penalty, LASSO shrinks less important variable coefficients to exactly zero, thereby automatically eliminating irrelevant or redundant predictors. This is particularly advantageous in our dataset, which includes a large number of physiological, laboratory, treatment, and demographic variables, many of which may be collinear or non-informative. Unlike models such as K-Nearest Neighbors or Support Vector Machines, which lack built-in feature selection mechanisms and treat all input variables equally, LASSO produces a sparse, interpretable model that highlights the most predictive features—such as BMI or SOFA scores—while reducing noise. This makes it an ideal choice for identifying key mortality risk factors and improving downstream model performance.

```r
cv_lasso <- cv.glmnet(X, y, alpha=1, family="binomial")
lasso_model <- glmnet(X, y, alpha=1, family="binomial", lambda=cv_lasso$lambda.min)

# Selected features
coef(lasso_model)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"
##                                          s0
## (Intercept)                    1.1816896942
## person_id                      .
## gender_concept_id8507          .
## gender_concept_id8532          .
## race_concept_id8515            .
## race_concept_id8516            .
## race_concept_id8527            .
## race_concept_id8657            .
## ethnicity_concept_id38003563 -0.7595093348
## ethnicity_concept_id38003564   .
## age_at_admission               .
## ecmo_duration_days             .
## bmi_avg                        .
## creatinine_avg                 .
## ferritin_avg                   .
## crp_avg                        .
## neutrophil_count_avg           .
## lymphocyte_count_avg           .
## neutrophil_lymphocyte_ratio    0.0649809949
## pao2_avg                       .
## fio2_avg                       .
## pao2_fio2_ratio                .
## platelets_avg                 -0.0003657619
## bilirubin_avg                  .
## systolic_bp_avg                .
## diastolic_bp_avg              -0.0227083889
## map_avg                        .
## sofa_respiratory_score         .
## sofa_coagulation_score         0.4232435663
## sofa_liver_score               .
## sofa_cardiovascular_score      .
## sofa_total_score               0.0343792217
## sofa_components_available      .
## received_norepinephrine        .
## received_vasopressin           .
## received_epinephrine           .
## received_angiotensin_ii        .
## received_any_vasopressor       .
## received_succinylcholine       .
## received_rocuronium            .
## received_vecuronium           .
## received_cisatracurium        .
## received_hydrocortisone       .
## received_methylprednisolone   .
## received_dexamethasone        .
## received_heparin              .
## received_fentanyl             0.0304606679
## received_propofol             .
## received_enoxaparin           .
## received_crrt                 .
## obese                         .
```

```
selected_features <- rownames(coef(lasso_model))[which(coef(lasso_model)!=0)][-1] # excluding intercept
print(selected_features)
```

```
## [1] "ethnicity_concept_id38003563" "neutrophil_lymphocyte_ratio"
## [3] "platelets_avg"                "diastolic_bp_avg"
## [5] "sofa_coagulation_score"       "sofa_total_score"
## [7] "received_fentanyl"
```

Prepare Dataset for 5-Fold Cross Validation

In this study, 5-fold cross-validation was used to ensure robust and generalizable evaluation of mortality prediction models for patients on VV-ECMO. Given the moderate sample size and high dimensionality of clinical data—including demographic, physiological, and treatment variables—5-fold cross-validation provides a reliable estimate of each model's performance while minimizing overfitting. By partitioning the data into five equal subsets, the model is trained on 80% of the data and tested on the remaining 20% in each iteration, ensuring that every observation is used for both training and validation. This approach balances computational efficiency with the need for stability in performance metrics such as AUC and accuracy, making it especially suitable for healthcare applications where predictive reliability is critical. Additionally, using 5-fold cross-validation allows for fair comparison across different algorithms under consistent resampling conditions.

```
# Prepare final dataset with selected features
df_model <- df[,c("race_concept_id", "ethnicity_concept_id",
                  selected_features[-c(1,2,3)], "died")]
train_control <- trainControl(method="cv", number=5, classProbs=TRUE, summaryFunction=twoClassSummary)

# Convert outcome to factor for caret
df_model$died <- factor(ifelse(df_model$died==1, "Died", "Survived"), levels=c("Survived", "Died"))

# Store models and results
models_list <- list()
results <- data.frame()
```

Models

In this study, we employed a diverse set of predictive modeling approaches— **logistic regression, random forest, gradient boosting machine (GBM), XGBoost**—to model in-hospital mortality among patients receiving VV-ECMO, with a particular focus on the role of obesity. Each model represents a different class of algorithm, offering complementary strengths in handling high-dimensional, nonlinear, and heterogeneous clinical data.

Logistic regression is a widely used statistical model for binary outcomes such as mortality. It estimates the probability of an event occurring (death) based on a linear combination of input variables. One key advantage of logistic regression is its interpretability—odds ratios provide clinically meaningful estimates of the association between each predictor (e.g., obesity, SOFA score) and the outcome. In this study, logistic regression allows us to directly quantify the adjusted effect of obesity on mortality, while controlling for other covariates such as age, organ dysfunction, and treatment indicators.

Decision trees offer a simple yet powerful method for classification by recursively splitting the data based on variables that best separate the outcome classes. They are intuitive and easy to visualize, making them useful for clinical interpretation. However, single trees can be unstable and prone to overfitting. To overcome this, we include random forest, an ensemble of decision trees that reduces variance by averaging predictions across many bootstrapped trees. Random forest also provides a robust estimate of variable importance, which is helpful for understanding which clinical features most strongly influence mortality.

Gradient boosting machine (GBM) and XGBoost are more sophisticated tree-based ensemble methods that build trees sequentially, each one correcting errors from the previous. These models typically outperform random forests in terms of predictive accuracy, especially in structured tabular data like ours. XGBoost, in particular, incorporates regularization, missing data handling, and efficient computation, making it a popular choice in machine learning competitions and healthcare research. In this VV-ECMO cohort, where interactions between variables (e.g., BMI and vasopressor use) may be complex, boosting methods help capture nonlinear and higher-order patterns.

By including these diverse models, we aim to balance interpretability and predictive power. This comprehensive modeling approach allows us to evaluate the robustness of the obesity-mortality relationship across different algorithmic perspectives and identify the best-performing models for clinical prediction.

```
##################################################
# Models
##################################################
# Create 10 folds
folds <- cut(seq(1, nrow(df_model)), breaks = 10, labels = FALSE)

# Initialize AUC storage
auc_logit <- c()
auc_rf    <- c()
auc_gbm   <- c()
auc_xgb   <- c()

# Add before CV loop
roc_list_logit <- list()
roc_list_rf <- list()
roc_list_gbm <- list()
roc_list_xgb <- list()


for (i in 1:10) {

  cat("Fold", i, "\n")

  # Split data
  test_idx <- which(folds == i)
  train_data <- df_model[-test_idx, ]
  test_data  <- df_model[test_idx, ]

  ### Logistic Regression ###
  logit_model <- glm(died ~ ., data = train_data, family = binomial())
  logit_prob <- predict(logit_model, newdata = test_data, type = "response")
  logit_roc <- roc(test_data$died, logit_prob)
  auc_logit[i] <- auc(logit_roc)
  # Inside the loop (after computing `logit_roc`)
  roc_list_logit[[i]] <- logit_roc

  ### Random Forest ###
  rf_model <- randomForest(died ~ ., data = train_data, ntree = 500)
  rf_prob <- predict(rf_model, newdata = test_data, type = "prob")[, "Died"]
  rf_roc <- roc(test_data$died, rf_prob)
  auc_rf[i] <- auc(rf_roc)
  # Inside the loop (after computing `logit_roc`)
  roc_list_rf[[i]] <- rf_roc

  ### GBM ###
  train_gbm <- train_data %>% mutate(y = ifelse(died == "Died", 1, 0)) %>% select(-died)
  test_gbm  <- test_data %>% mutate(y = ifelse(died == "Died", 1, 0)) %>% select(-died)
  gbm_model <- gbm(y ~ ., data = train_gbm,
                   distribution = "bernoulli",
                   n.trees = 100, interaction.depth = 3,
                   shrinkage = 0.05, verbose = FALSE)
  gbm_prob <- predict(gbm_model, newdata = test_gbm, n.trees = 100, type = "response")
  gbm_roc <- roc(test_gbm$y, gbm_prob)
  auc_gbm[i] <- auc(gbm_roc)
  # Inside the loop (after computing `logit_roc`)
  roc_list_gbm[[i]] <- gbm_roc

  ### XGBoost ###
  X_train <- model.matrix(died ~ . -1, data = train_data)
  y_train <- ifelse(train_data$died == "Died", 1, 0)
  X_test  <- model.matrix(died ~ . -1, data = test_data)
  y_test  <- ifelse(test_data$died == "Died", 1, 0)

  xgb_model <- xgboost(data = X_train, label = y_train,
                       objective = "binary:logistic",
                       eval_metric = "auc",
                       nrounds = 100, verbose = 0)
  xgb_prob <- predict(xgb_model, newdata = X_test)
  xgb_roc <- roc(y_test, xgb_prob)
  auc_xgb[i] <- auc(xgb_roc)
  roc_list_xgb[[i]] <- xgb_roc
}
```

```
## Fold 1

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 2

## Setting levels: control = Survived, case = Died
## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 3

## Setting levels: control = Survived, case = Died
## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 4

## Setting levels: control = Survived, case = Died

## Setting direction: controls > cases
## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls > cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 5

## Setting levels: control = Survived, case = Died

## Setting direction: controls > cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls > cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 6

## Setting levels: control = Survived, case = Died
## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 7
## Setting levels: control = Survived, case = Died
## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Fold 8

## Setting levels: control = Survived, case = Died

## Setting direction: controls > cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls > cases

## Setting levels: control = 0, case = 1

## Setting direction: controls > cases

## Setting levels: control = 0, case = 1

## Setting direction: controls > cases

## Fold 9

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases

## Fold 10

## Setting levels: control = Survived, case = Died
## Setting direction: controls < cases

## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

Model Comparisons

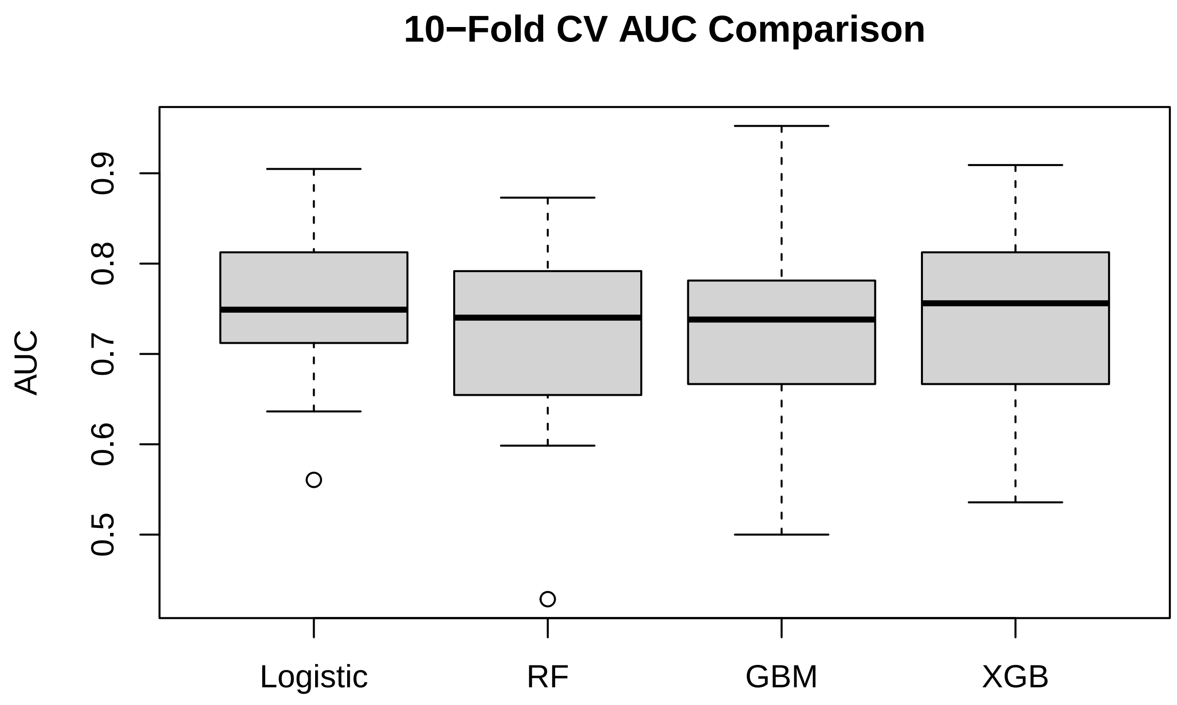
```
auc_table_cv <- data.frame(
  Model = c("Logistic Regression", "Random Forest", "GBM", "XGBoost"),
  Mean_AUC = round(c(mean(auc_logit), mean(auc_rf), mean(auc_gbm), mean(auc_xgb)), 3),
  Median_AUC = round(c(median(auc_logit), median(auc_rf), median(auc_gbm), median(auc_xgb)), 3),
  SD_AUC = round(c(sd(auc_logit), sd(auc_rf), sd(auc_gbm), sd(auc_xgb)), 3)
)

print(auc_table_cv)
```

```
##                 Model Mean_AUC Median_AUC SD_AUC
## 1 Logistic Regression    0.751      0.749  0.104
## 2       Random Forest    0.705      0.740  0.125
## 3                 GBM    0.733      0.738  0.122
## 4             XGBoost    0.735      0.756  0.109
```

Visualization

```
boxplot(auc_logit, auc_rf, auc_gbm, auc_xgb,
        names = c("Logistic", "RF", "GBM", "XGB"),
        ylab = "AUC", main = "10-Fold CV AUC Comparison")
```



```r
# Optionally add smoothed average ROC (interpolate)
avg_roc_logit <- roc(response = unlist(lapply(roc_list_logit, `[[`, "response")),
               predictor = unlist(lapply(roc_list_logit, `[[`, "predictor")))
```

```
## Setting levels: control = Survived, case = Died

## Setting direction: controls < cases
```

```r
avg_roc_gbm <- roc(response = unlist(lapply(roc_list_gbm, `[[`, "response")),
                     predictor = unlist(lapply(roc_list_gbm, `[[`, "predictor")))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

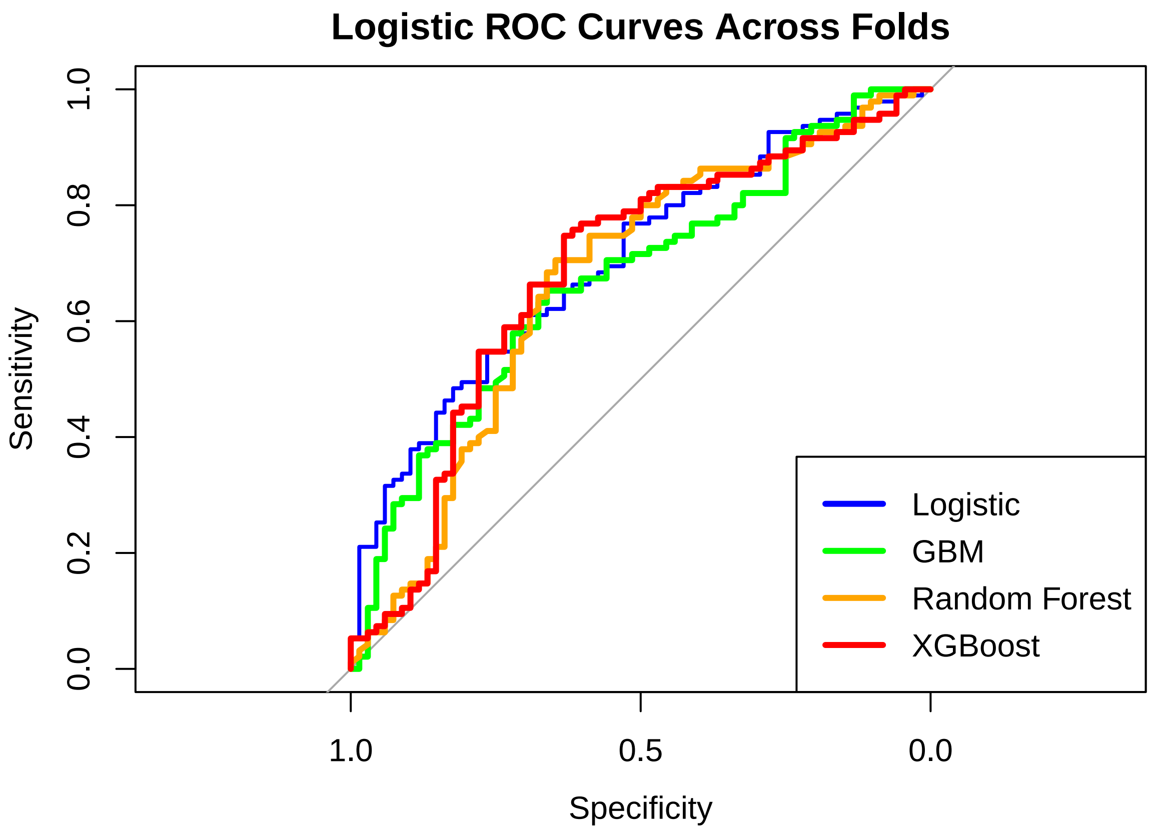
```r
avg_roc_rf <- roc(response = unlist(lapply(roc_list_rf, `[[`, "response")),
                   predictor = unlist(lapply(roc_list_rf, `[[`, "predictor")))
```

```
## Setting levels: control = Survived, case = Died
## Setting direction: controls < cases
```

```r
avg_roc_xgb <- roc(response = unlist(lapply(roc_list_xgb, `[[`, "response")),
                   predictor = unlist(lapply(roc_list_xgb, `[[`, "predictor")))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
plot(avg_roc_logit, col = "blue", main="Logistic ROC Curves Across Folds")
lines(avg_roc_gbm, col = "green", lwd=3)
lines(avg_roc_rf, col = "orange", lwd=3)
lines(avg_roc_xgb, col = "red", lwd=3)
legend("bottomright",legend=c("Logistic","GBM", "Random Forest",  "XGBoost"),
       col=c("blue", "green", "orange", "red"),lwd=3)
```


Logistic ROC Curves Across Folds
Sensitivity
Specificity
Logistic
GBM
Random Forest
XGBoost