```
> source("E:/Education/Year 3 Spring Semester/STA3005/Assignments/Project/CounterStrikeWinAnalysis/R/Model Fitting and Tuning.R", echo=TRUE)

> # 导入所需的 R 包
> library(dplyr)

> library(readxl)

> library(caret)

> library(randomForest)

> library(e1071)

> library(xgboost)

> library(ggplot2)

> library(stringr)

> # ==============================
> # 1. 导入所需的库
> # ==============================
>
> # 这里导入的包包括数据处理、模型训练、评估等所需要的库
> # dplyr：用于数据处理
> # readxl：读取 Excel 文 .... [TRUNCATED]

> head(data_1)
```

```
# A tibble: 6 × 9
  `team    a`   de_mirage      `58.86486486486486`    `0.7263078236130868`    `1720.4`
`47.28921568627451` `0.7926147058823529` `1536`
  <chr>   <chr>              <dbl>           <dbl>   <dbl>          <dbl>          <dbl>
<dbl>
1 team a   de_inferno          52.3           0.765   1990.          54.8          0.763
2073
2 team b   de_vertigo          20.2           0.773    754.          66.1          0.723
158
3 team b   de_mirage           48.4           0.733   2565           62.6          0.838
2219
4 team b   de_mirage           43.6           0.710   1484.          55.5          0.769
1352
5 team b   de_anubis           29.8           0.596   1443.          45.3          0.691
1632
6 team a   de_anubis           78.4           0.71    1478.          41.7          0.738
1402
# ℹ 1 more variable: `1-051c5a18-6a99-4e5e-bef7-ed1143474b33` <chr>

> colnames(data_1) <- c("win", "map", "Team_A_avg_win_percentage", "Team_A_avg_KR",
"Team_A_avg_elo",
+                    "Team_B_avg_win_percenta ..." ... [TRUNCATED]

> # 2.2 读取 `data_2`、`data_3`、`data_4`、`data_5`、`data_6` 数据
> # 读取 data_2 数据
> data_2 <- read_excel("data_win_prediction_2.xlsx")

> colnames(data_2) <- c("win", "map", "Team_A_avg_win_percentage", "Team_A_avg_KR",
"Team_A_avg_elo",
+                    "Team_B_avg_win_percenta ..." ... [TRUNCATED]

> col_name <- colnames(data_2)[1]
```

```
> data_2 <- data_2[data_2[[1]] != col_name, ]

> head(data_2)
# A tibble: 6 × 9
  win    map         Team_A_avg_win_percentage Team_A_avg_KR    Team_A_avg_elo Team_B_avg_win_perce…¹ Team_B_avg_KR Team_B_avg_elo `Match ID`
  <chr>  <chr>       <chr>                     <chr>            <chr>          <chr>                  <chr>         <chr>          <chr>
1 team b de_inferno  40.50700280112044         0.7153239028944… 1881.4         47.11111111111111      0.6793111111… 1596           1-0d63019…
2 team a de_ancient  57.94378253325622         0.7428943204164… 1384           51.77777777777779      0.7095861111… 1375           1-d5c2d88…
3 team a de_dust2    48.26680672268908         0.67393038579068 1936.2         55.92746518897415      0.8015577223… 1884           1-fc6886c…
4 team a de_mirage   46.61409526589792         0.7194531017369… 1581.8         51.52439811418168      0.7252486311… 1562           1-9403e94…
5 team b de_anubis   42.02564102564103         0.7393974358974… 1066.8         54.45707070707071      0.6671079545… 831            1-9f2213d…
6 team b de_anubis   46.23850574712644         0.6986655172413… 2233           48.30774222070106      0.7249268221… 2323           1-5ae5261…
# ℹ abbreviated name: ¹Team_B_avg_win_percentage

> # 确保 data_2 中的列与 data_1 完全一致
> data_2 <- data_2 %>%
+   mutate(across(c("win", "map", "Match ID"), as.character)) %>%
+   mutate(across(c("Team_A_av ..." ... [TRUNCATED]

> data_3 <- read_excel("data_win_prediction_3.xlsx")

> head(data_3)
# A tibble: 6 × 9
```

```
  `team b` de_nuke `38.08941058941058` `0.812422077922078` `1231.2` `49.42982456140351` `0.7192236842105265` `1092` 1-b7867490-f764-456e…¹
  <chr>  <chr>      <dbl>     <dbl>  <dbl>      <dbl>      <dbl> <dbl> <chr>
1 team a de_mirage   62.4     0.834  1488.      50.1      0.715  1349 1-9433dec6-0ec3-4a2f-…
2 team a de_mirage   49.3     0.727  1387.      47.2      0.677  1428 1-29b163a3-be56-4bbb-…
3 team b de_dust2    41.2     0.669  1536.      62.8      0.738  1382 1-992a1ee2-e49c-47b2-…
4 team a de_anubis   50.6     0.720  1199.      42        0.665  1087 1-857e07fb-ee62-47d6-…
5 team a de_dust2    57.5     0.822  1340.      40.9      0.658  1277 1-66efc827-307f-4303-…
6 team b de_infer…   64.0     0.727  1859.      52.1      0.757  1846 1-28d7b4ca-d593-4dad-…
# ℹ abbreviated name: ¹`1-b7867490-f764-456e-b05c-0172094a75df`

> colnames(data_3) <- c("win", "map", "Team_A_avg_win_percentage", "Team_A_avg_KR", "Team_A_avg_elo",
+                       "Team_B_avg_win_percenta ..." ... [TRUNCATED]

> data_4 <- read_excel("data_win_prediction_4.xlsx")

> head(data_4)
# A tibble: 6 × 9
  `team b` de_dust2 `57.80952380952381` `0.7796571428571429` `1383.4` `48.73355629877369` `0.7092021181716834` `1437` 1-3996e84d-6a28-4234…¹
  <chr>  <chr>      <dbl>     <dbl>  <dbl>      <dbl>      <dbl> <dbl> <chr>
1 team b de_infe…    48.0     0.745  1465.      60.3      0.765  1382 1-f93179cc-2830-4f76-…
2 team a de_anub…    53.2     0.732  1802.      52.4      0.748
```

```
1700 1-57c128cf-dfd9-40a1-…

3 team b   de_anub…              60.6              0.732    2837.              50.1              0.733
3201 1-92513cc9-0ac2-4d02-…

4 team a   de_mira…              72.6              0.738    1161.              43.0              0.732
1040 1-f2c9de2f-9d8e-4754-…

5 team b   de_anci…              66.9              0.787    2466.              63.5              0.784
2147 1-f2d3fd92-4a47-42d5-…

6 team a   de_mira…              61.8              0.823    1348.              42.9              0.716
1186 1-cd501b0b-3908-4a23-…

# ℹ abbreviated name: ¹`1-3996e84d-6a28-4234-a65c-6b6ce3e802a8`


> colnames(data_4) <- c("win", "map", "Team_A_avg_win_percentage", "Team_A_avg_KR", "Team_A_avg_elo",

+                        "Team_B_avg_win_percenta ..." ... [TRUNCATED]


> data_5 <- read_excel("data_win_prediction_5.xlsx")


> head(data_5)

# A tibble: 6 × 9

  `team b`  de_mirage  `52.24772824772825`  `0.7487128466128465`  `1636.2` `54.45553848779655` `0.7786094896740059` `1669`

  <chr>    <chr>              <dbl>             <dbl>    <dbl>             <dbl>             <dbl> <dbl>

1 team b   de_mirage          44.5              0.693    1353.              59.4              0.832
1449

2 team b   de_mirage          47.9              0.734    1251.              50.6              0.688
1220

3 team a   de_mirage          54.5              0.734    1876.              54.4              0.787
1655

4 team b   de_dust2           30.7              0.681    1955.              49.7              0.727
2186

5 team b   de_nuke            38                0.853    1441.              62.5              0.697
1270
```

```
6 team a   de_mirage              52.8              0.728    1902.             55.6              0.676
1733
# ℹ 1 more variable: `1-e60841fa-8284-42bd-99da-e119eb65692c` <chr>

> colnames(data_5) <- c("win", "map", "Team_A_avg_win_percentage", "Team_A_avg_KR",
"Team_A_avg_elo",
+                      "Team_B_avg_win_percenta ..." ... [TRUNCATED]

> data_6 <- read_excel("data_win_prediction_6.xlsx")

> head(data_6)
# A tibble: 6 × 9
  `team    a`   de_inferno   `68.06959706959707`   `0.7589274725274725`   `1558.6`
`53.19668737060041` `0.7445536231884058` `1637`
  <chr>   <chr>              <dbl>             <dbl>   <dbl>             <dbl>             <dbl>
<dbl>
1 team a   de_anubis              61.3              0.753    1757              64.0              0.717
1871
2 team a   de_ancient             51.9              0.677    1609.             44.7              0.763
1756
3 team b   de_mirage              50.5              0.708    1435.             51.2              0.759
1604
4 team a   de_nuke                52.0              0.735    1838.             28                0.649
1646
5 team a   de_train               66.9              0.767    2179.             20                0.780
2040
6 team a   de_mirage              51.6              0.707    1774.             55.4              0.766
1501
# ℹ 1 more variable: `1-ececcce7-0859-4b9f-ba5a-f3f5bb343feb` <chr>

> colnames(data_6) <- c("win", "map", "Team_A_avg_win_percentage", "Team_A_avg_KR",
"Team_A_avg_elo",
+                      "Team_B_avg_win_percenta ..." ... [TRUNCATED]
```

```
> # ==============================
> # 3. 数据清理
> # ==============================
>
> # 3.1 合并所有数据集
> data_combined <- bind_rows(data_1, data_2, data_3,  .... [TRUNCATED]

> # 3.2 过滤不以“de”开头的地图
> data_combined <- data_combined %>% filter(str_starts(map, "de"))

> # 3.3 过滤出至少 50 场比赛的地图
> filtered_maps <- data_combined %>% count(map) %>% filter(n >= 50) %>% pull(map)

> data_combined <- data_combined %>% filter(map %in% filtered_maps)

> # 3.4 删除低 Elo 评分的比赛
> data_combined <- data_combined %>% filter(Team_A_avg_elo > 800 &
Team_B_avg_elo > 800)

> # 3.5 检查缺失数据
> sum(is.na(data_combined))
[1] 12

> # 3.6 删除缺失数据
> data_combined <- na.omit(data_combined)

> # 3.7 删除重复数据
> data_combined <- distinct(data_combined)
```

```
> # 3.8 保存清理后的数据
> write.csv(data_combined, "win_prediction_data_clean.csv", row.names = FALSE)

> # ============================
> # 4. 特征工程
> # ============================
>
> # 4.1 创建虚拟变量（正确方式）
> data_encoded <- data_combined %>%
+   select(- .... [TRUNCATED]

> # 用 dummyVars 正确处理虚拟变量
> dummy <- dummyVars(win ~ ., data = data_encoded)

> features <- predict(dummy, newdata = data_encoded) %>% as.data.frame()

> # 4.2 定义 response（唯一正确的定义）
> response <- data_encoded$win  # 必须是 factor

> # ============================
> # 5. 逻辑回归模型
> # ============================
>
> # 5.1 交叉验证
> train_control <- trainControl(method = "cv", number  .... [TRUNCATED]

> # 5.2 训练逻辑回归模型
> lr_model <- train(features, response, method = "glm", trControl = train_control, family
```

```
= "binomial")

> # 5.3 输出模型的准确度
> lr_accuracy <- lr_model$results$Accuracy

> print(lr_accuracy)
[1] 0.7676761

> # 5.4 计算准确度的置信区间
> ci <- (1.96 * sqrt((lr_accuracy * (1 - lr_accuracy)) / 8104)) * 100

> print(paste("Confidence Interval:", round(ci, 2)))
[1] "Confidence Interval: 0.92"

> # ==============================
> # 6. 随机森林模型（正确版）
> # ==============================
>
> # 6.1 数据拆分
> set.seed(50)

> train_index <- createDataPartition(response, p = 0.7, list = FALSE)

> X_train <- features[train_index, ]

> X_test <- features[-train_index, ]

> y_train <- response[train_index]  # response 此时已确定为 factor
```

```
> y_test <- response[-train_index]

> # 强制检查 y_train 是否是因子
> y_train <- factor(y_train)

> stopifnot(is.factor(y_train))

> # 检查维度一致性
> stopifnot(nrow(X_train) == length(y_train))

> # 检查 y_train 的类别数
> print(length(unique(y_train)))
[1] 2

> # 6.2 训练随机森林模型
> rf_model <- randomForest(
+   x = X_train,
+   y = y_train,  # 已确认是 factor
+   ntree = 500,
+   importance = TRUE
+ )

> # 输出模型信息
> print(rf_model)

Call:
 randomForest(x = X_train, y = y_train, ntree = 500, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
```

```
No. of variables tried at each split: 3

        OOB estimate of  error rate: 22.73%
Confusion matrix:
       team a team b class.error
team a   2631    727   0.2164979
team b    808   2587   0.2379971

> # 6.3 预测并评估准确度
> rf_pred <- predict(rf_model, X_test)

> rf_accuracy <- mean(rf_pred == y_test)  # 计算预测准确度

> print(paste("Accuracy:", round(rf_accuracy, 4)))
[1] "Accuracy: 0.7743"

> # 6.4 随机森林的超参数调优
> param_grid <- expand.grid(mtry = c(2, 3, 4))

> rf_tune <- train(
+   x = X_train,
+   y = y_train,
+   method = "rf",
+   trControl = trainControl(method = "cv", number = 10),
+   tuneGrid = para .... [TRUNCATED]

> print(rf_tune$bestTune)
  mtry
3    4
```

```
> # ==============================
> # 7. 支持向量机 (SVM)
> # ==============================
>
> # 7.1 训练基本 SVM 模型
> svm_model <- svm(x = X_train, y = y_train, .... [TRUNCATED]

> # 7.2 预测并评估准确度
> svm_pred <- predict(svm_model, X_test)

> svm_accuracy <- mean(svm_pred == y_test)

> print(paste("SVM Base Accuracy:", round(svm_accuracy, 4)))
[1] "SVM Base Accuracy: 0.7694"

> # 7.3 SVM 超参数调优 - 修复参数网格
> # 定义正确的参数网格，与 svmRadial 方法兼容
> param_grid_svm <- expand.grid(
+   sigma = c(0.1, 0.01, 1),  # 对应 Python 中的 gamma
+   C = c(0.1, .... [TRUNCATED]

> # 执行网格搜索
> set.seed(50)

> svm_tune <- train(
+   x = X_train,
+   y = y_train,
+   method = "svmRadial",  # 使用径向基核函数(对应 Python 中的'rbf')
```

```
+   trControl = trainControl(method = "c ..." ... [TRUNCATED]

> # 输出最佳参数
> print("SVM Best Parameters:")
[1] "SVM Best Parameters:"

> print(svm_tune$bestTune)
  sigma C
2  0.01 1

> # 7.4 使用最佳参数重新拟合 SVM 模型
> best_svm_model <- svm(
+   x = X_train,
+   y = y_train,
+   kernel = "radial",  # 对应 Python 中的'rbf'
+   cost = svm_tune$bestT .... [TRUNCATED]

> # 预测并评估准确度
> best_svm_pred <- predict(best_svm_model, X_test)

> best_svm_accuracy <- mean(best_svm_pred == y_test)

> print(paste("SVM Best Model Accuracy:", round(best_svm_accuracy, 4)))
[1] "SVM Best Model Accuracy: 0.7712"

> # 7.5 计算 SVM 模型准确率的置信区间
> svm_ci <- (1.96 * sqrt((best_svm_accuracy * (1 - best_svm_accuracy)) / length(y_test))) * 100
```

```
> print(paste("SVM Confidence Interval:", round(svm_ci, 2)))
[1] "SVM Confidence Interval: 1.53"

> # =============================
> # 8. XGBoost 模型
> # =============================
>
> # 8.1 转换数据格式，XGBoost 要求标签为 0/1 整数
> # 确保标签编码为 0/1
> # 将因子标签转换为 0/1 数 .... [TRUNCATED]

> table(response_encoded)  # 检查编码是否正确
response_encoded
   0    1
4797 4849

> # 分割数据
> set.seed(50)

> xgb_train_index <- createDataPartition(response_encoded, p = 0.7, list = FALSE)

> xgb_X_train <- as.matrix(features[xgb_train_index, ])

> xgb_X_test <- as.matrix(features[-xgb_train_index, ])

> xgb_y_train <- response_encoded[xgb_train_index]

> xgb_y_test <- response_encoded[-xgb_train_index]
```

```
> # 创建 DMatrix 对象
> dtrain <- xgb.DMatrix(data = xgb_X_train, label = xgb_y_train)

> dtest <- xgb.DMatrix(data = xgb_X_test, label = xgb_y_test)

> # 8.2 设置基本参数
> params <- list(
+   objective = "binary:logistic",
+   max_depth = 3,
+   eta = 0.1,
+   gamma = 0,
+   subsample = 1,
+   colsample_ .... [TRUNCATED]

> # 8.3 训练基本 XGBoost 模型
> xgb_model <- xgb.train(
+   params = params,
+   data = dtrain,
+   nrounds = 10,
+   watchlist = list(train = dtrain),
+   ve .... [TRUNCATED]

> # 预测并评估基本模型
> xgb_pred <- predict(xgb_model, dtest)

> xgb_pred_binary <- ifelse(xgb_pred > 0.5, 1, 0)

> xgb_accuracy <- mean(xgb_pred_binary == xgb_y_test)
```

```
> print(paste("XGBoost Baseline Accuracy:", round(xgb_accuracy, 4)))
[1] "XGBoost Baseline Accuracy: 0.7636"

> # 8.4 绘制特征重要性
> importance_matrix <- xgb.importance(model = xgb_model)

> print(importance_matrix)
                     Feature        Gain       Cover  Frequency
                      <char>       <num>       <num>      <num>
1: Team_A_avg_win_percentage 0.600681168 0.526393332 0.48571429
2: Team_B_avg_win_percentage 0.397542846 0.465940545 0.50000000
3:            Team_A_avg_elo 0.001775985 0.007666124 0.01428571

> xgb.plot.importance(importance_matrix, top_n = 10)

> # 8.5 交叉验证找最佳轮数
> cv_results <- xgb.cv(
+   params = params,
+   data = dtrain,
+   nrounds = 40,
+   nfold = 3,
+   early_stopping_rounds = 10,
+   .... [TRUNCATED]

> # 输出最佳轮数
> best_nrounds <- which.min(cv_results$evaluation_log$test_logloss_mean)

> print(paste("Best number of rounds:", best_nrounds))
[1] "Best number of rounds: 40"
```

```
> # 8.6 超参数调优 - 使用随机参数网格而不是完整网格
> # 定义参数采样范围 - 类似 Python 的 RandomizedSearchCV
> set.seed(123)

> param_combinations <- list(
+   max_depth = sample(3:11, 4),
+   eta = sample(c(0.001, 0.01, 0.1, 0.2, 0.3), 3),
+   gamma = sample(c(0, 0.1, 0.5, 1 .... [TRUNCATED]

> # 初始化最佳参数和分数
> best_accuracy <- 0

> best_params <- NULL

> best_nrounds <- 25  # 使用之前发现的最佳轮数

> # 创建参数网格
> param_grid <- expand.grid(
+   max_depth = param_combinations$max_depth,
+   eta = param_combinations$eta,
+   gamma = param_combinations .... [TRUNCATED]

> # 随机抽样 12 个组合
> sampled_indices <- sample(1:nrow(param_grid), min(12, nrow(param_grid)))

> sampled_params <- param_grid[sampled_indices, ]

> # 对每个采样的参数组合进行评估
```

```
> for (i in 1:nrow(sampled_params)) {
+   current_params <- list(
+     objective = "binary:logistic",
+     max_depth = sampled_pa .... [TRUNCATED]

> # 输出最佳参数和准确率
> print(paste("Best XGBoost CV Accuracy:", round(best_accuracy, 4)))
[1] "Best XGBoost CV Accuracy: 0.7693"

> print("Best XGBoost Parameters:")
[1] "Best XGBoost Parameters:"

> print(best_params)
$objective
[1] "binary:logistic"

$max_depth
[1] 11

$eta
[1] 0.01

$gamma
[1] 0.1

$subsample
[1] 0.5

$colsample_bytree
```

```
[1] 0.75

$min_child_weight
[1] 5

$lambda
[1] 1

$alpha
[1] 0.1


> # 8.7 使用最佳参数训练最终模型
> final_xgb_model <- xgb.train(
+   params = best_params,
+   data = dtrain,
+   nrounds = 25,
+   watchlist = list(train = dtrai .... [TRUNCATED]

> # 预测并评估最终模型
> final_xgb_pred <- predict(final_xgb_model, dtest)

> final_xgb_pred_binary <- ifelse(final_xgb_pred > 0.5, 1, 0)

> final_xgb_accuracy <- mean(final_xgb_pred_binary == xgb_y_test)

> print(paste("Final XGBoost Accuracy:", round(final_xgb_accuracy, 4)))
[1] "Final XGBoost Accuracy: 0.7611"
```

```
> # 8.8 模型变量重要性可视化
> final_importance <- xgb.importance(model = final_xgb_model)

> print(final_importance)
                       Feature         Gain       Cover   Frequency
                        <char>        <num>       <num>       <num>
 1: Team_A_avg_win_percentage 0.4338500414 0.275717450 0.181224900
 2: Team_B_avg_win_percentage 0.3866251330 0.331350189 0.222389558
 3:              Team_A_avg_KR 0.0584541238 0.122477500 0.149598394
 4:              Team_B_avg_KR 0.0535900104 0.115847707 0.172188755
 5:             Team_A_avg_elo 0.0347776550 0.076755248 0.134538153
 6:             Team_B_avg_elo 0.0259934437 0.054055393 0.107429719
 7:               mapde_mirage 0.0026962496 0.006331561 0.014056225
 8:              mapde_inferno 0.0013255882 0.005344963 0.005020080
 9:                mapde_dust2 0.0011074867 0.003907450 0.005020080
10:               mapde_anubis 0.0006476188 0.001581079 0.002510040
11:              mapde_vertigo 0.0005842891 0.005542192 0.004518072
12:              mapde_ancient 0.0003483602 0.001089267 0.001506024

> xgb.plot.importance(final_importance, top_n = 10)

> # 8.9 输出最终 XGBoost 模型的所有准确率指标
> conf_matrix <- table(Predicted = final_xgb_pred_binary, Actual = xgb_y_test)

> print("Confusion Matrix:")
[1] "Confusion Matrix:"

> print(conf_matrix)
         Actual
```

```
Predicted    0    1
        0 1112  374
        1  317 1090

> precision <- conf_matrix[2,2] / sum(conf_matrix[2,])

> recall <- conf_matrix[2,2] / sum(conf_matrix[,2])

> f1_score <- 2 * precision * recall / (precision + recall)

> print(paste("Precision:", round(precision, 4)))
[1] "Precision: 0.7747"

> print(paste("Recall:", round(recall, 4)))
[1] "Recall: 0.7445"

> print(paste("F1 Score:", round(f1_score, 4)))
[1] "F1 Score: 0.7593"

> # ============================
> # 9. 模型比较
> # ============================
>
> # 9.1 汇总各模型准确率（确保变量名完全匹配）
> model_accuracies <- data.frame(
+   Mod .... [TRUNCATED]

> # 9.2 可视化（添加置信区间和评估类型说明）
> library(scales)  # 确保加载 scales 包用于百分比格式
```

```
> ggplot(model_accuracies, aes(x = reorder(Model, -Accuracy),
+                              y = Accuracy,
+                              fill = Model .... [TRUNCATED]

> # 9.3 输出关键统计信息
> cat("\n=== 模型评估关键指标 ===\n")

=== 模型评估关键指标 ===

> cat(sprintf("逻辑回归置信区间: ±%.2f%%\n", ci))
逻辑回归置信区间: ±0.92%

> cat(sprintf("随机森林测试样本量: %d\n", length(y_test)))
随机森林测试样本量: 2893

> cat(sprintf("XGBoost F1 分数: %.3f\n", f1_score))
XGBoost F1 分数: 0.759
警告信息:
1: In model.frame.default(Terms, newdata, na.action = na.action, xlev = object$lvls) :
  变量'win'不是因子
2: Removed 4 rows containing missing values or values outside the scale range (`geom_bar()`).
```