SCRIP TFM 2025

Pedro Ojeda Soto

02 de junio de 2025

Índice

1. Carga de datos y preprocesamientos

```r
#  Cargar librerías necesarias
library(dplyr)
library(ggplot2)
library(caret)

#  Cargar el dataset
df <- read.csv("D:/2024/UOC/TFM/survey lung cancer.csv", header = TRUE, sep = ",")

#  Revisar estructura de los datos
str(df)

#  Convertir variables categóricas a factores
df$GENDER <- as.factor(df$GENDER)
df$LUNG_CANCER <- as.factor(ifelse(df$LUNG_CANCER == "YES", 1, 0))  # Convertir a binario

#  Verificar valores faltantes
missing_values <- colSums(is.na(df))
print("Valores faltantes por variable:")
print(missing_values)

#  Reemplazar valores faltantes con la moda (para categóricas) y con la mediana (para numéricas)
Mode <- function(x) { ux <- unique(x); ux[which.max(tabulate(match(x, ux)))] }
for (col in colnames(df)) {
  if (is.numeric(df[[col]])) {
    df[[col]][is.na(df[[col]])] <- median(df[[col]], na.rm = TRUE)
  } else {
    df[[col]][is.na(df[[col]])] <- as.character(Mode(df[[col]]))
  }
}

#  Normalización y estandarización de variables numéricas
num_vars <- names(df)[sapply(df, is.numeric)]
df[, num_vars] <- scale(df[, num_vars])

#  Guardar el dataset preprocesado
write.csv(df, "survey_lung_cancer_preprocessed.csv", row.names = FALSE)
```

2. Visualización exploratoria

```r
# Cargar librerías

library(ggplot2)
library(dplyr)
library(readr)
library(patchwork)

# Cargar dataset
df <- read_csv("D:/2024/UOC/TFM/survey lung cancer.csv")

# Limpiar nombres de columnas
colnames(df) <- gsub(" ", "_", colnames(df))
colnames(df) <- gsub("\\s+$", "", colnames(df)) # quitar espacios finales

# Convertir variable de interés en factor
df$LUNG_CANCER <- factor(df$LUNG_CANCER, levels = c("NO", "YES"))

# Agrupar la edad en rangos para análisis categórico
df$AGE_GROUP <- cut(df$AGE, breaks = c(0, 40, 50, 60, 70, 100),
                    labels = c("0-40", "41-50", "51-60", "61-70", "71+"))

# Detectar variables binarias numéricas que pueden considerarse categóricas
numericas <- names(df)[sapply(df, is.numeric)]
binarias <- numericas[sapply(df[numericas], function(x) all(x %in% c(0, 1, 2)))]

# Convertir esas binarias a factor
df[binarias] <- lapply(df[binarias], factor)

# Crear lista final de variables categóricas
categoricas_final <- c("GENDER", "AGE_GROUP", binarias)

# Reetiquetar factores binarios 1 = "NO", 2 = "SI"
for (var in binarias) {
  df[[var]] <- factor(df[[var]],
                      levels = c(1, 2),
                      labels = c("NO", "SI"))
}

# Función para graficar variables categóricas
grafico_cat <- function(var) {
  ggplot(df, aes_string(x = var, fill = "LUNG_CANCER")) +
    geom_bar(position = "fill") +
    labs(title = paste("Distribución de", var), y = "Proporción", x = NULL) +
    theme_minimal() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 9),
      axis.text.x = element_text(size = 8, angle = 45, hjust = 1),
      axis.text.y = element_text(size = 8)
    )
}

# Crear lista de gráficos
```

```r
plots_cat <- lapply(categoricas_final, grafico_cat)

# Mostrar en grillas de 2 por fila
for (i in seq(1, length(plots_cat), by = 2)) {
  p1 <- plots_cat[[i]]
  p2 <- if (i + 1 <= length(plots_cat)) plots_cat[[i + 1]] else NULL

  if (!is.null(p2)) {
    print(p1 + p2)
  } else {
    print(p1)
  }
}
```

3. Box-plot Genero vs Cancer de Pulmón

```r
# Cargar librerías
library(ggplot2)
library(dplyr)
library(readr)

# Cargar dataset
df <- read_csv("survey lung cancer.csv")

# Limpiar nombres de columnas
colnames(df) <- gsub(" ", "_", colnames(df))
colnames(df) <- gsub("\\s+$", "", colnames(df))

# Convertir a factores
df$LUNG_CANCER <- factor(df$LUNG_CANCER, levels = c("NO", "YES"))
df$GENDER <- factor(df$GENDER, levels = c("F", "M"), labels = c("Femenino", "Masculino"))

# Crear boxplot: Edad vs Sexo, coloreado por Cáncer
ggplot(df, aes(x = GENDER, y = AGE, fill = LUNG_CANCER)) +
  geom_boxplot(position = position_dodge(0.8), width = 0.6, outlier.shape = 21, outlier.size = 1.5) +
  labs(title = "Distribución de edad por sexo y cáncer de pulmón",
       x = "Sexo",
       y = "Edad",
       fill = "Cáncer de pulmón") +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 14, hjust = 0.5),
    axis.text = element_text(size = 10),
    axis.title = element_text(size = 12),
    legend.position = "top"
  ) +
  scale_fill_manual(values = c("NO" = "#00BFC4", "YES" = "#F8766D"))
```

4. Modelos Regresión Logística, Random Forest y XGboost con validación cruzada y Umsampling

```r
# Cargar librerías
library(caret)
library(pROC)
library(dplyr)
library(randomForest)
library(xgboost)

# 1. Cargar datos
df <- read.csv("D:/2024/UOC/TFM/survey_lung_cancer_preprocessed.csv")
df$LUNG_CANCER <- factor(df$LUNG_CANCER, levels = c(0,1), labels = c("No", "Yes"))

# 2. Configurar validación cruzada estratificada con upsampling
set.seed(12345)
ctrl <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final",
  sampling = "up"
)

# 3. Función para obtener métricas
obtener_metricas <- function(pred_df) {
  cm <- confusionMatrix(pred_df$pred, pred_df$obs, positive = "Yes")
  auc_val <- auc(roc(pred_df$obs, pred_df$Yes))
  data.frame(
    Accuracy = cm$overall["Accuracy"],
    Sensibilidad = cm$byClass["Sensitivity"],
    Especificidad = cm$byClass["Specificity"],
    Balanced_Accuracy = cm$byClass["Balanced Accuracy"],
    F1_score = cm$byClass["F1"],
    AUC = auc_val
  )
}

# 4. Regresión Logística
log_model_cv <- train(
  LUNG_CANCER ~ ., data = df,
  method = "glm",
  family = "binomial",
  trControl = ctrl,
  metric = "ROC"
)
metricas_log <- obtener_metricas(log_model_cv$pred)
roc_log_cv <- roc(response = log_model_cv$pred$obs, predictor = log_model_cv$pred$Yes)

# 5. Random Forest
rf_model_cv <- train(
  LUNG_CANCER ~ ., data = df,
  method = "rf",
```

```
  trControl = ctrl,
  metric = "ROC"
)
rf_pred <- rf_model_cv$pred
best_mtry <- rf_model_cv$bestTune$mtry
rf_pred <- rf_pred[rf_pred$mtry == best_mtry, ]
metricas_rf <- obtener_metricas(rf_pred)
roc_rf_cv <- roc(response = rf_pred$obs, predictor = rf_pred$Yes)

# 6. XGBoost
df_xgb <- df %>%
  mutate(across(where(is.character), as.factor)) %>%
  mutate(across(setdiff(names(.)[sapply(., is.factor)], "LUNG_CANCER"), as.numeric))

xgb_model_cv <- train(
  LUNG_CANCER ~ ., data = df_xgb,
  method = "xgbTree",
  trControl = ctrl,
  metric = "ROC"
)
xgb_pred <- xgb_model_cv$pred
best_param <- xgb_model_cv$bestTune
for (param in names(best_param)) {
  xgb_pred <- xgb_pred[xgb_pred[[param]] == best_param[[param]], ]
}
metricas_xgb <- obtener_metricas(xgb_pred)
roc_xgb_cv <- roc(response = xgb_pred$obs, predictor = xgb_pred$Yes)

# 7. Unir métricas en una tabla
metricas_finales <- rbind(
  cbind(Modelo = "Regresión Logística", metricas_log),
  cbind(Modelo = "Random Forest", metricas_rf),
  cbind(Modelo = "XGBoost", metricas_xgb)
)
print(metricas_finales)

# 8. Graficar todas las curvas ROC
plot(roc_log_cv, col = "green", lwd = 2, main = "Curvas ROC - Comparación de Modelos")
plot(roc_rf_cv, col = "blue", lwd = 2, add = TRUE)
plot(roc_xgb_cv, col = "red", lwd = 2, add = TRUE)
legend("bottomright", legend = c("Regresión Logística", "Random Forest", "XGBoost"),
       col = c("green", "blue", "red"), lwd = 2)
```

```
# Extraer valores AUC redondeados
auc_log <- round(auc(roc_log_cv), 2)
auc_rf <- round(auc(roc_rf_cv), 2)
auc_xgb <- round(auc(roc_xgb_cv), 2)


# 9. Importancia de variables (opcional)
plot(varImp(log_model_cv), main = "Importancia - Regresión Logística")
```

```
plot(varImp(rf_model_cv), main = "Importancia - Random Forest")
```

```r
plot(varImp(xgb_model_cv), main = "Importancia - XGBoost")
```

5. Modelos Regresión Logística con validación cruzada y Umsampling

```r
# Cargar librerías necesarias
library(caret)
library(dplyr)
library(pROC)

# Función para calcular métricas completas desde las predicciones
obtener_metricas <- function(pred_df) {
  cm <- confusionMatrix(pred_df$pred, pred_df$obs, positive = "Yes")
  auc_val <- auc(roc(pred_df$obs, pred_df$Yes))

  data.frame(
    Accuracy = cm$overall["Accuracy"],
    Sensibilidad = cm$byClass["Sensitivity"],
    Especificidad = cm$byClass["Specificity"],
    Balanced_Accuracy = cm$byClass["Balanced Accuracy"],
    F1_score = cm$byClass["F1"],
    AUC = auc_val
  )
}

# Cargar datos
df <- read.csv("D:/2024/UOC/TFM/survey_lung_cancer_preprocessed.csv")

# Asegurar que la variable de salida sea factor con nombres válidos
df$LUNG_CANCER <- factor(df$LUNG_CANCER, levels = c(0,1), labels = c("No", "Yes"))

# Configurar validación cruzada estratificada con UPSAMPLING
set.seed(12345)
ctrl <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final",
  sampling = "up"
)

# Entrenar modelo con validación cruzada
log_model_cv <- train(
  LUNG_CANCER ~ .,
  data = df,
  method = "glm",
  family = "binomial",
  trControl = ctrl,
  metric = "ROC"
)

# Mostrar resumen del modelo entrenado por caret
print(log_model_cv)

# Extraer el modelo final
final_model <- log_model_cv$finalModel
```

```r
# Calcular odds ratios
odds_ratios <- exp(coef(final_model))
confint_odds <- exp(confint(final_model))

# Imprimir odds ratios e intervalos
print("Odds Ratios:")
print(odds_ratios)

print("Intervalos de confianza al 95%:")
print(confint_odds)

# Curva ROC con validación cruzada
roc_log_cv <- roc(response = log_model_cv$pred$obs,
                  predictor = log_model_cv$pred$Yes)
auc_log_cv <- auc(roc_log_cv)
print(paste("AUC promedio (CV) - Regresión Logística:", round(auc_log_cv, 3)))

# Graficar curva ROC
plot(roc_log_cv, col = "darkgreen", lwd = 2,
     main = "Curva ROC - Regresión Logística (Upsampling)")
```

```r
# Cálculo de métricas finales
metricas_log <- obtener_metricas(log_model_cv$pred)
print(metricas_log)

# Importancia de variables
log_importance <- varImp(log_model_cv)
plot(log_importance, main = "Importancia de Variables - Regresión Logística")
```

6. Modelo Random Forest con validación cruzada y Umsampling

```r
library(caret)
library(dplyr)
library(pROC)
library(randomForest)

# Cargar datos
df <- read.csv("D:/2024/UOC/TFM/survey_lung_cancer_preprocessed.csv")
df$LUNG_CANCER <- factor(df$LUNG_CANCER, levels = c(0,1), labels = c("No", "Yes"))

# Validación cruzada estratificada con upsampling
set.seed(12345)
ctrl <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final",
  sampling = "up"
)

# Entrenar modelo
rf_model_cv <- train(
  LUNG_CANCER ~ .,
  data = df,
  method = "rf",
  trControl = ctrl,
  metric = "ROC"
)

# Mostrar resultados
print(rf_model_cv)

# Curva ROC
roc_rf_cv <- roc(response = rf_model_cv$pred$obs,
                 predictor = rf_model_cv$pred$Yes)
plot(roc_rf_cv, col = "blue", main = "Curva ROC - Random Forest (Upsampling)")
```

```r
# Métricas
obtener_metricas <- function(pred_df) {
  cm <- confusionMatrix(pred_df$pred, pred_df$obs, positive = "Yes")
  auc_val <- auc(roc(pred_df$obs, pred_df$Yes))
  data.frame(
    Accuracy = cm$overall["Accuracy"],
    Sensibilidad = cm$byClass["Sensitivity"],
    Especificidad = cm$byClass["Specificity"],
    Balanced_Accuracy = cm$byClass["Balanced Accuracy"],
    F1_score = cm$byClass["F1"],
    AUC = auc_val
  )
}
metricas_rf <- obtener_metricas(rf_model_cv$pred)
print(metricas_rf)
```

```r
# Importancia de variables
plot(varImp(rf_model_cv), main = "Importancia de Variables - Random Forest")
```

7. Modelo XGboost con validación cruzada y Umsampling

```
library(caret)
library(dplyr)
library(pROC)
library(xgboost)

# Cargar datos
df <- read.csv("D:/2024/UOC/TFM/survey_lung_cancer_preprocessed.csv")
df$LUNG_CANCER <- factor(df$LUNG_CANCER, levels = c(0,1), labels = c("No", "Yes"))

# Conversión de variables categóricas
df_xgb <- df %>%
  mutate(across(where(is.character), as.factor)) %>%
  mutate(across(
    .cols = setdiff(names(.)[sapply(., is.factor)], "LUNG_CANCER"),
    .fns = as.numeric
  ))

# Validación cruzada estratificada con upsampling
set.seed(12345)
ctrl <- trainControl(
  method = "cv",
  number = 10,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  savePredictions = "final",
  sampling = "up"
)

# Entrenar modelo
xgb_model_cv <- train(
  LUNG_CANCER ~ .,
  data = df_xgb,
  method = "xgbTree",
  trControl = ctrl,
  metric = "ROC"
)

# Mostrar resultados
print(xgb_model_cv)

# Curva ROC
roc_xgb_cv <- roc(response = xgb_model_cv$pred$obs,
                  predictor = xgb_model_cv$pred$Yes)
plot(roc_xgb_cv, col = "red", main = "Curva ROC - XGBoost (Upsampling)")
```

```
# Métricas
metricas_xgb <- obtener_metricas(xgb_model_cv$pred)
print(metricas_xgb)

# Importancia de variables
plot(varImp(xgb_model_cv), main = "Importancia de Variables - XGBoost")
```

8. Intervalos de Confianza para las Métricas

```r
# Cargar librerías necesarias
library(caret)
library(dplyr)
library(pROC)

# Función F1
F_meas <- function(pred, obs, relevant = "Yes") {
  cm <- confusionMatrix(pred, obs, positive = relevant)
  precision <- cm$byClass["Precision"]
  recall <- cm$byClass["Sensitivity"]
  if ((precision + recall) == 0) return(0)
  return(2 * precision * recall / (precision + recall))
}

# Función que calcula todas las métricas por fold
metricas_fold_completas <- function(pred_df) {
  pred_df %>%
    group_by(Resample) %>%
    summarise(
      AUC = as.numeric(auc(roc(obs, Yes))),
      Accuracy = mean(pred == obs),
      Sensibilidad = sensitivity(pred, obs, positive = "Yes"),
      Especificidad = specificity(pred, obs, negative = "No"),
      Balanced_Accuracy = (Sensibilidad + Especificidad)/2,
      F1 = F_meas(pred, obs, relevant = "Yes")
    )
}

# Aplicamos la función a cada modelo
folds_log <- metricas_fold_completas(log_model_cv$pred)
folds_rf  <- metricas_fold_completas(rf_pred)
folds_xgb <- metricas_fold_completas(xgb_pred)

# Función resumen  IC 95%
resumen_ic_completo <- function(folds_df, modelo) {
  data.frame(
    Modelo = modelo,
    AUC = sprintf("%.3f [%.3f-%.3f]",
                  mean(folds_df$AUC),
                  t.test(folds_df$AUC)$conf.int[1],
                  t.test(folds_df$AUC)$conf.int[2]),
    Accuracy = sprintf("%.3f [%.3f-%.3f]",
                       mean(folds_df$Accuracy),
                       t.test(folds_df$Accuracy)$conf.int[1],
                       t.test(folds_df$Accuracy)$conf.int[2]),
    Sensibilidad = sprintf("%.3f [%.3f-%.3f]",
                           mean(folds_df$Sensibilidad),
                           t.test(folds_df$Sensibilidad)$conf.int[1],
                           t.test(folds_df$Sensibilidad)$conf.int[2]),
    Especificidad = sprintf("%.3f [%.3f-%.3f]",
                            mean(folds_df$Especificidad),
                            t.test(folds_df$Especificidad)$conf.int[1],
```

```
                            t.test(folds_df$Especificidad)$conf.int[2]),
    Balanced_Accuracy = sprintf("%.3f [%.3f-%.3f]",
                                mean(folds_df$Balanced_Accuracy),
                                t.test(folds_df$Balanced_Accuracy)$conf.int[1],
                                t.test(folds_df$Balanced_Accuracy)$conf.int[2]),
    F1 = sprintf("%.3f [%.3f-%.3f]",
                 mean(folds_df$F1),
                 t.test(folds_df$F1)$conf.int[1],
                 t.test(folds_df$F1)$conf.int[2])
  )
}

# Tabla completa
tabla_ic_completa <- bind_rows(
  resumen_ic_completo(folds_log, "Regresión Logística"),
  resumen_ic_completo(folds_rf, "Random Forest"),
  resumen_ic_completo(folds_xgb, "XGBoost")
)

print(tabla_ic_completa)
```

9. Importancia de Variables para Modelo Logístico

```r
# Cargar librerías necesarias
library(caret)
library(ggplot2)

# Entrenar el modelo logístico con caret
modelo_log <- train(
  LUNG_CANCER ~ ., data = df,
  method = "glm",
  family = "binomial",
  trControl = trainControl(method = "cv", number = 10)
)

# Extraer importancia de variables
imp <- varImp(modelo_log)$importance
imp$Variable <- rownames(imp)

# Graficar con ggplot2
ggplot(imp, aes(x = reorder(Variable, Overall), y = Overall)) +
  geom_col(fill = "#0072B2") +
  coord_flip() +
  labs(
    title = "Importancia de Variables - Regresión Logística",
    x = "Variable",
    y = "Importancia"
  ) +
  theme_minimal(base_size = 10)
```

10. Importancia de Variables para Random Forest

```r
# Cargar librerías necesarias
library(caret)
library(ggplot2)

# Obtener importancia de variables desde el modelo Random Forest
imp_rf <- varImp(rf_model_cv)$importance

# Agregar los nombres de variables como columna
imp_rf$Variable <- rownames(imp_rf)

# Ordenar por importancia
imp_rf <- imp_rf[order(imp_rf$Overall, decreasing = TRUE), ]

# Gráfico con ggplot2 en formato de barras horizontales
ggplot(imp_rf, aes(x = reorder(Variable, Overall), y = Overall)) +
  geom_col(fill = "#0072B2") +
  coord_flip() +
  labs(
    title = "Importancia de Variables - Random Forest",
    x = "Variable",
    y = "Importancia"
  ) +
```

```
  theme_minimal(base_size = 11)
```

11. Importancia de Variables para XgBoost

```
# Cargar librerías
library(caret)
library(ggplot2)

# Obtener importancia de variables
imp_xgb <- varImp(xgb_model_cv)$importance

# Agregar columna de nombres
imp_xgb$Variable <- rownames(imp_xgb)

# Ordenar por importancia descendente
imp_xgb <- imp_xgb[order(imp_xgb$Overall, decreasing = TRUE), ]

# Graficar
ggplot(imp_xgb, aes(x = Overall, y = reorder(Variable, Overall))) +
  geom_point(color = "#0072B2", size = 3) +
  geom_segment(aes(x = 0, xend = Overall, y = Variable, yend = Variable), color = "#0072B2") +
  labs(
    title = "Importancia - XGBoost",
    x = "Importancia",
    y = NULL
  ) +
  theme_minimal(base_size = 11)
```

12. Matriz de confusión para Regresión Logística

```
library(caret)
cm_log <- confusionMatrix(log_model_cv$pred$pred, log_model_cv$pred$obs, positive = "Yes")
print(cm_log)
```

13. Matriz de confusión para Random Forest

```
cm_rf <- confusionMatrix(rf_model_cv$pred$pred, rf_model_cv$pred$obs, positive = "Yes")
print(cm_rf)
```

14. Matriz de confusión para XgBoost

```
cm_xgb <- confusionMatrix(xgb_model_cv$pred$pred, xgb_model_cv$pred$obs, positive = "Yes")
print(cm_xgb)
```

14. Intervalo de Confianza para las Métricas

```
# Cargar librerías necesarias
library(caret)
library(dplyr)
library(pROC)
```

```r
# Función F1
F_meas <- function(pred, obs, relevant = "Yes") {
  cm <- confusionMatrix(pred, obs, positive = relevant)
  precision <- cm$byClass["Precision"]
  recall <- cm$byClass["Sensitivity"]
  if ((precision + recall) == 0) return(0)
  return(2 * precision * recall / (precision + recall))
}

# Función que calcula todas las métricas por fold
metricas_fold_completas <- function(pred_df) {
  pred_df %>%
    group_by(Resample) %>%
    summarise(
      AUC = as.numeric(auc(roc(obs, Yes))),
      Accuracy = mean(pred == obs),
      Sensibilidad = sensitivity(pred, obs, positive = "Yes"),
      Especificidad = specificity(pred, obs, negative = "No"),
      Balanced_Accuracy = (Sensibilidad + Especificidad)/2,
      F1 = F_meas(pred, obs, relevant = "Yes")
    )
}

# Aplicamos la función a cada modelo
folds_log <- metricas_fold_completas(log_model_cv$pred)
folds_rf  <- metricas_fold_completas(rf_pred)
folds_xgb <- metricas_fold_completas(xgb_pred)

# Función resumen  IC 95%
resumen_ic_completo <- function(folds_df, modelo) {
  data.frame(
    Modelo = modelo,
    AUC = sprintf("%.3f [%.3f-%.3f]",
                  mean(folds_df$AUC),
                  t.test(folds_df$AUC)$conf.int[1],
                  t.test(folds_df$AUC)$conf.int[2]),
    Accuracy = sprintf("%.3f [%.3f-%.3f]",
                       mean(folds_df$Accuracy),
                       t.test(folds_df$Accuracy)$conf.int[1],
                       t.test(folds_df$Accuracy)$conf.int[2]),
    Sensibilidad = sprintf("%.3f [%.3f-%.3f]",
                           mean(folds_df$Sensibilidad),
                           t.test(folds_df$Sensibilidad)$conf.int[1],
                           t.test(folds_df$Sensibilidad)$conf.int[2]),
    Especificidad = sprintf("%.3f [%.3f-%.3f]",
                            mean(folds_df$Especificidad),
                            t.test(folds_df$Especificidad)$conf.int[1],
                            t.test(folds_df$Especificidad)$conf.int[2]),
    Balanced_Accuracy = sprintf("%.3f [%.3f-%.3f]",
                                mean(folds_df$Balanced_Accuracy),
                                t.test(folds_df$Balanced_Accuracy)$conf.int[1],
                                t.test(folds_df$Balanced_Accuracy)$conf.int[2]),
    F1 = sprintf("%.3f [%.3f-%.3f]",
```

```
                mean(folds_df$F1),
                t.test(folds_df$F1)$conf.int[1],
                t.test(folds_df$F1)$conf.int[2])
  )
}

# Tabla completa
tabla_ic_completa <- bind_rows(
  resumen_ic_completo(folds_log, "Regresión Logística"),
  resumen_ic_completo(folds_rf, "Random Forest"),
  resumen_ic_completo(folds_xgb, "XGBoost")
)

print(tabla_ic_completa)
```