Informe del Modelo: Random Forest con Hiperparámetros

Descripción del Modelo

El segundo modelo se construyó sobre la base de Random Forest, aplicando ajustes de hiperparámetros para intentar mejorar su desempeño en la predicción de fallos en el proceso de fabricación de chips. Este enfoque buscó optimizar el modelo básico utilizando técnicas de validación cruzada y experimentación con diferentes configuraciones clave.

Hiperparámetros Ajustados

- **Número de variables consideradas por división (`mtry`):** Probado con valores de 2, 5 y 10.
- **Criterio de división (`splitrule`):** `Gini impurity`.
- **Tamaño mínimo de nodos (`min.node.size`):** Probado con valores de 1, 5 y 10.

Propósito

Este script busca mejorar el rendimiento del modelo predictivo para clasificar los estados **Pass/Fail** del conjunto de datos SECOM ajustando hiperparámetros del modelo **Random Forest**. Además, evalúa la necesidad de probar otros enfoques como **XGBoost** debido a los resultados sub-óptimos alcanzados.

Pasos Detallados

---

1. Carga del Dataset

```
secom_data <- read.csv("C:/SECOM_Analysis/data/uci_secom.csv")
dim(secom_data)
colnames(secom_data)
head(secom_data)
```

Propósito:

- Cargar los datos y confirmar que están estructurados correctamente.

Por qué:

- La validación inicial del set de datos asegura que las siguientes etapas se puedan realizar sin errores.

---

2. Limpieza de Datos

Manejo de valores faltantes:

```
secom_data <- secom_data[, colMeans(is.na(secom_data)) < 0.5]
secom_data <- secom_data %>%
  mutate(across(everything(), ~ ifelse(is.na(.), median(., na.rm = TRUE), .)))
```

Propósito:

- Eliminar columnas con más del 50% de datos faltantes e imputar valores faltantes con la mediana.

Por qué:

- Reducir el impacto de datos faltantes en el modelo mientras se preserva la mayoría de las características relevantes.

---

3. Normalización y Reducción de la Dimensión

Normalización:

```
scaled_data <- scale(secom_data[, -c(1, ncol(secom_data))])
```

Propósito:

- Escalar los datos para que las características estén en el mismo rango.

Reducción de Dimensiones:

```
pca_result <- prcomp(scaled_data, center = TRUE, scale. = TRUE)
```

Propósito:

- Usar PCA para identificar las componentes principales que explican la mayor parte de la varianza.

Ajustes posibles:

- El número de componentes principales retenidos puede ajustarse para optimizar el equilibrio entre precisión y complejidad.

---

4. Balanceo de Clases

```
balanced_failures <- failures[sample(nrow(failures), nrow(successes), replace
= TRUE), ]
balanced_data <- rbind(successes, balanced_failures)
```

Propósito:

- Equilibrar las clases `Pass` y `Fail` mediante sobremuestreo de la clase minoritaria.

Por qué:

- Los datos desbalanceados sesgan el modelo hacia la clase mayoritaria, lo que reduce la precisión en la clase minoritaria.

---

5. Ajuste de Hiperparámetros con Random Forest

Definir cuadrícula de hiperparámetros:

```
tune_grid <- expand.grid(
  mtry = c(2, 5, 10),
  splitrule = "gini",
  min.node.size = c(1, 5, 10)
)
```

Propósito:

- Probar combinaciones de hiperparámetros como el número de variables (`mtry`) y tamaño mínimo de nodos (`min.node.size`).

Reentrenar modelo optimizado:

```
rf_tuned <- randomForest(
  Pass.Fail ~ .,
  data = train_data,
  ntree = 500,
  mtry = sqrt(ncol(train_data) - 1),
  importance = TRUE
)
```

Propósito:

- Ajustar el modelo con los mejores parámetros y evaluar su rendimiento.

---

6. Evaluación del Modelo

Matriz de Confusión:

```
conf_matrix <- confusionMatrix(rf_predictions, test_data$Pass.Fail)
```

Métricas:

```
accuracy <- conf_matrix$overall["Accuracy"]
precision <- conf_matrix$byClass["Precision"]
recall <- conf_matrix$byClass["Recall"]
specificity <- conf_matrix$byClass["Specificity"]
```

Propósito:

- Calcular métricas clave como precisión, sensibilidad y especificidad.

---

7. Visualización de Resultados

Curva ROC:

```
roc_curve <- roc(test_data$Pass.Fail, rf_prob, levels = c("Fail", "Pass"))
plot(roc_curve, main = "Curva ROC", col = "blue")
```

Propósito:

- Evaluar la capacidad del modelo para distinguir entre clases mediante el área bajo la curva AUC. Se refiere al área bajo la curva ROC (Receiver Operating Characteristic), que es una métrica utilizada para evaluar el rendimiento de un modelo de clasificación binaria.

Importancia de Variables:

```
var_imp <- varImp(rf_tuned)
plot(var_imp, main = "Importancia de Variables")
```

Propósito:

- Identificar las características que más contribuyen a la clasificación.

Resultados

- **Matriz de Confusión:**

```
          Predicted
```

```
     Fail Pass
Fail   14    6
Pass    7   13
```

- **Métricas obtenidas:**
  - **Precisión (Accuracy):** 67.5%.
  - **Precisión Positiva (Precision):** 66.7%.
  - **Sensibilidad (Recall):** 70%.
  - **Especificidad:** 65%.
  - **AUC (Área Bajo la Curva):** 0.72 en promedio.

Análisis

1. **Comparación con el modelo básico:**
   - La precisión, sensibilidad y especificidad no mostraron mejoras significativas. De hecho, algunos parámetros, como la precisión (67.5%) y la especificidad (65%), tuvieron un ligero empeoramiento en comparación con el modelo básico.
   - Este resultado sugiere que el ajuste de hiperparámetros, en este caso, no fue suficiente para superar las limitaciones del modelo inicial.
2. **Limitaciones identificadas:**
   - El desempeño estancado podría deberse a la alta dimensionalidad del dataset y la posible redundancia entre características.
   - Aunque Random Forest es robusto, la naturaleza de los datos puede requerir modelos más sofisticados o técnicas adicionales de selección de características.

Aunque el ajuste de hiperparámetros es una técnica valiosa, en este caso, no se logró una mejora significativa respecto al modelo básico. Esto nos llevó a considerar un enfoque alternativo mediante el uso de **XGBoost**, un modelo de Gradient Boosting que puede manejar datos de alta dimensionalidad de manera más eficiente y aprovechar relaciones complejas entre las características.

Recomendación

Para proyectos similares donde los ajustes en modelos estándar no generan mejoras claras, se recomienda:

1. Probar algoritmos más avanzados, como **XGBoost** o **Gradient Boosted Machines**.
2. Reducir la dimensionalidad con técnicas como PCA o selección de características basada en importancia.
3. Continuar iterando para optimizar tanto la precisión como la generalización del modelo.

El modelo con **XGBoost**, que analizaremos en el siguiente paso, representará una evolución en este proyecto.

Conclusión

El modelo **Random Forest** alcanzó una precisión inicial del **70%** con su configuración predeterminada. Posteriormente, al ajustar los hiperparámetros (como `mtry` y `min.node.size`), la precisión disminuyó al **67%**. Este resultado indica que los ajustes realizados no mejoraron el desempeño del modelo, posiblemente debido a limitaciones en las características del dataset o a una estructura compleja de los datos.

Dada la disminución en el rendimiento, se decidió explorar el modelo **XGBoost**, conocido por su capacidad de manejar conjuntos de datos complejos y características correlacionadas. Este cambio tiene el potencial de superar las limitaciones observadas con Random Forest y mejorar la precisión en la clasificación.