Informe del Modelo Básico: Random Forest

Descripción del Modelo

El modelo básico utiliza **Random Forest**, una técnica de aprendizaje automático basada en árboles de decisión. Random Forest es ideal para tareas de clasificación, como la predicción de fallos en el proceso de fabricación de chips, ya que:

- Combina múltiples árboles de decisión para mejorar la precisión.
- Reduce el riesgo de sobreajuste (overfitting).
- Es robusto frente a datos faltantes y ruidos.

Conjunto de Datos

- **Dataset:** SECOM (Sistema de Monitorización de Condiciones) proveniente de sensores en una línea de producción de chips.
- **Dimensiones iniciales:** 1567 filas y 590 columnas.
- **Limpieza aplicada:**
    - Se eliminaron columnas con más del 50% de valores faltantes.
    - Los valores faltantes restantes se imputaron con la mediana.
    - Se eliminaron columnas con varianza cero, resultando en 562 columnas finales.

Metodología

1. **Balanceo de Clases:**
    - El dataset original estaba desbalanceado en términos de etiquetas "Pass" (aprobado) y "Fail" (fallido).
    - Se realizó sobremuestreo de la clase minoritaria para equilibrar las proporciones.
2. **División de Datos:**
    - **80%** para entrenamiento.
    - **20%** para prueba.
3. **Entrenamiento del Modelo:**
    - Se utilizó Random Forest con los parámetros por defecto:
        - Número de árboles: 500.
        - Métrica de división: `Gini impurity`.
        - Selección de variables: Aleatoria en cada división.

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 el dataset bruto desde un archivo CSV y verificar su contenido.

Por qué:

- Garantizar que los datos estén disponibles y estructurados correctamente antes de proceder.

Ajustes posibles:

- Si el archivo tiene diferentes delimitadores (e.g., `;`), añade `sep=";"` al comando `read.csv`.

---

2. Limpieza de Datos

Identificar y manejar valores faltantes:

```
na_counts <- colSums(is.na(secom_data))
na_percentage <- colMeans(is.na(secom_data)) * 100
secom_data <- secom_data[, colMeans(is.na(secom_data)) < 0.5]
```

Propósito:

- Eliminar columnas con más del 50% de datos faltantes y garantizar que las columnas restantes sean completas.

Por qué:

- Las columnas con demasiados datos faltantes son poco útiles para el análisis y podrían distorsionar el modelo.

Ajustes posibles:

- Cambiar el umbral del 50% según la tolerancia al manejo de valores faltantes.

Imputación de valores faltantes:

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

Propósito:

- Rellenar valores faltantes con la mediana de cada columna para evitar sesgos.

Por qué:

- La mediana es menos sensible a valores atípicos en comparación con el promedio.

---

3. Normalización y Escalado

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

Propósito:

- Escalar los datos para garantizar que todas las variables tengan el mismo rango.

Por qué:

- Modelos como Random Forest y PCA son sensibles a las escalas de las variables.

Ajustes posibles:

- Usar otros métodos de normalización, como `min-max`, si la distribución de los datos lo requiere.

---

4. Reducción de la Dimensión

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

Propósito:

- Usar Análisis de Componentes Principales (PCA) para reducir la cantidad de variables al tiempo que se conserva la mayor parte de la información.

Por qué:

- Simplifica el modelo al reducir las variables redundantes.

Ajustes posibles:

- Cambiar el número de componentes retenidos en función de la varianza explicada.

---

5. Equilibrio de Clases

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

Propósito:

- Sobremuestrear la clase minoritaria para equilibrar el conjunto de datos.

Por qué:

- Los modelos entrenados en datos desbalanceados tienden a favorecer la clase mayoritaria.

Ajustes posibles:

- Usa técnicas como SMOTE si el sobremuestreo no es suficiente.

---

6. Modelo Predictivo (Random Forest)

```
rf_model <- randomForest(Pass.Fail ~ ., data = train_data, importance = TRUE)
rf_predictions <- predict(rf_model, test_data)
```

Propósito:

- Entrenar un modelo de Random Forest para predecir el estado Pass/Fail.

Por qué:

- Random Forest maneja bien datos complejos y ruidosos, y proporciona métricas de importancia de características.

Ajustes posibles:

- Ajustar el número de árboles (`ntree`) o el número de variables por división (`mtry`) para mejorar el rendimiento.

---

7. Evaluación del Modelo

```
true_positive <- conf_matrix["Pass", "Pass"]
true_negative <- conf_matrix["Fail", "Fail"]
accuracy <- (true_positive + true_negative) / sum(conf_matrix)
```

Propósito:

- Calcular métricas como precisión, sensibilidad y especificidad para evaluar el desempeño del modelo.

Por qué:

- Estas métricas ofrecen una visión completa del rendimiento del modelo.

Ajustes posibles:

- Cambiar el umbral para las predicciones probabilísticas para maximizar la métrica que más interese (ejemplo; sensibilidad).

---

8. Análisis Exploratorio (EDA)

```
barplot(
  table(secom_data$Pass.Fail),
  col = c("red", "green"),
  main = "Distribución de Pass/Fail"
)
```

Propósito:

- Visualizar la distribución de la variable objetivo.

Por qué:

- Detectar desbalances en las clases y generar insights iniciales.

---

9. Curva ROC y AUC

```
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 utilizando el área bajo la curva (AUC).

Por qué:

- AUC es una métrica robusta para problemas de clasificación binaria, especialmente en datos desbalanceados.

---

10. Visualización de la Importancia de las Variables

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

Propósito:

- Identificar las variables que más contribuyen al modelo.

Por qué:

- Ayuda a entender qué aspectos del proceso son más críticos para el resultado Pass/Fail.

Resultados

- **Matriz de Confusión:**

```
     Predicted
      Fail Pass
Fail   14    6
Pass    6   14
```

- **Métricas de Evaluación:**
  - **Precisión (Accuracy):** 70%.
  - **Precisión Positiva (Precision):** 70%.
  - **Sensibilidad (Recall):** 70%.
  - **Especificidad:** 70%.

Observaciones

- El modelo mostró un desempeño aceptable como línea base, con un 70% de precisión en la predicción de fallos y aprobaciones.
- La sensibilidad (70%) indica que el modelo detecta correctamente el 70% de los fallos reales, mientras que la especificidad (70%) muestra que el modelo identifica correctamente el 70% de los aprobados.
- Algunas limitaciones, como la baja especificidad, sugieren la necesidad de ajustes posteriores para mejorar la identificación de aprobados.

Recomendaciones

1. Optimizare el modelo con hiperparámetros del modelo para mejorar las métricas, especialmente la especificidad.
2. Explorar técnicas de selección de características para reducir dimensionalidad y mejorar eficiencia.
3. Considerare otros modelos, como Gradient Boosting o XGBoost, para comparar resultados.

Conclusión

Este modelo básico establece una línea base para el análisis predictivo en el proceso de fabricación de chips. Aunque su desempeño es moderado, proporciona una base sólida para iteraciones

posteriores y mejora continua, sin embargo, una mejora en la predicción es el objetivo a alcanzar por esta razón ajustaremos hiperparametros.